Form Methods Syst Des (2016) 49:33–74
DOI 10.1007/s10703-016-0241-z

ModelPlex: verified runtime validation of verified cyber-physical system models

Stefan Mitsch[1,2] · **André Platzer**[1]

Published online: 18 February 2016

Abstract Formal verification and validation play a crucial role in making cyber-physical systems (CPS) safe. Formal methods make strong guarantees about the system behavior *if* accurate models of the system can be obtained, including models of the controller and of the physical dynamics. In CPS, models are essential; but any model we could possibly build necessarily deviates from the real world. If the real system fits to the model, its behavior is guaranteed to satisfy the correctness properties verified with respect to the model. Otherwise, all bets are off. This article introduces ModelPlex, a method ensuring that verification results about models apply to CPS implementations. ModelPlex provides correctness guarantees for CPS executions at runtime: it combines offline verification of CPS models with runtime validation of system executions for compliance with the model. ModelPlex ensures in a provably correct way that the verification results obtained for the model apply to the actual system runs by monitoring the behavior of the world for compliance with the model. If, at some point, the observed behavior no longer complies with the model so that offline verification results no longer apply, ModelPlex initiates provably safe fallback actions, assuming the system dynamics deviation is bounded. This article, furthermore, develops a systematic technique to synthesize provably correct monitors automatically from CPS proofs in differential dynamic logic by a correct-by-construction approach, leading to *verifiably correct runtime model validation*. Overall, ModelPlex generates provably correct monitor conditions that, if checked to hold at runtime, are provably guaranteed to imply that the offline safety verification results about the CPS model apply to the present run of the actual CPS implementation.

Keywords Runtime verification · Static verification · Cyber-physical systems · Hybrid systems · Differential dynamic logic

✉ Stefan Mitsch
smitsch@cs.cmu.edu

André Platzer
aplatzer@cs.cmu.edu

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh, USA

[2] Present Address: Department of Cooperative Information Systems, Johannes Kepler University, Linz, Austria

1 Introduction

Cyber-physical systems (CPS) involve controllers and the relevant dynamics of the environment. Since safety is crucial for CPS, their models (e. g., hybrid system models [31]) need to be verified formally. Formal *verification* guarantees that a model is safe with respect to a safety property. The remaining task is to *validate* whether the model is adequate, so that the verification results for the model transfer to the actual system implementation [18,42]. This article introduces ModelPlex [24], a method to *synthesize correct-by-construction monitors for CPS by theorem proving automatically*: it uses sound axioms and proof rules of differential dynamic logic [33] to formally verify that a model is safe and to synthesize provably correct monitors that validate compliance of system executions with that model. The difficult question answered by ModelPlex is *what exact conditions need to be monitored* at runtime to guarantee compliance with the models and thus safety.

System execution, however, provides many opportunities for surprising deviations from the model: faults may cause the system to function improperly [43], sensors may deliver uncertain values, actuators may suffer from disturbance, or the formal verification may have assumed simpler ideal-world dynamics for tractability reasons or made unrealistically strong assumptions about the behavior of other agents in the environment. Simpler models are often better for time-critical decisions and optimizations, because they make it possible to compute predictions at the rate required for real-time decisions. The same phenomenon of simplicity for predictability is often exploited for the models in formal verification and validation, where formal verification results are often easier to obtain for simpler models. It is more helpful to obtain a verification or prediction result about a simpler model than to fail on a more complex one. The flipside is that the *verification results obtained about models of a CPS only apply to the actual CPS at runtime to the extent that the system fits to the model*. ModelPlex enables tradeoffs between analytic power and accuracy of models while retaining strong safety guarantees.

Validation, i. e., checking whether a CPS implementation fits to a model, is an interesting but difficult problem. Even more so, since CPS models are more difficult to analyze than ordinary (discrete) programs because of the continuous physical plant, the environment, sensor inaccuracies, and actuator disturbance, making full model validation quite elusive.

In this article, we, thus, settle for the question of *runtime model validation*, i. e. validating whether the model assumed for verification purposes is adequate for a *particular system execution* to ensure that the offline safety verification results apply *to the current execution*.[1] But we focus on *verifiably correct runtime validation* to ensure that verified properties of models provably apply to the CPS implementation, which is important for safety and certification [5]. Only with such a way of validating model compliance is there an unbroken chain of evidence of safety claims that apply to the actual system, rather than merely to its model. ModelPlex provides a chain of formal proofs as a strong form of such evidence.

At runtime, ModelPlex monitors check for model compliance. If the observed system execution fits to the verified model, then this execution is safe according to the offline verification result about the model. If it does not fit, then the system is potentially unsafe because it evolves outside the verified model and no longer has an applicable safety proof, so that a verified fail-safe action from the model is initiated to avoid safety risks, cf. Fig. 1. System-

[1] ModelPlex checks system execution w.r.t. a monitor specification, and thus, belongs to the field of runtime verification [18]. In this article we use the term *runtime validation* in order to clearly convey the purpose of monitoring (i. e., runtime verification monitors properties without offline verification; ModelPlex validates adequacy of models to transfer offline verification results to the online situation).

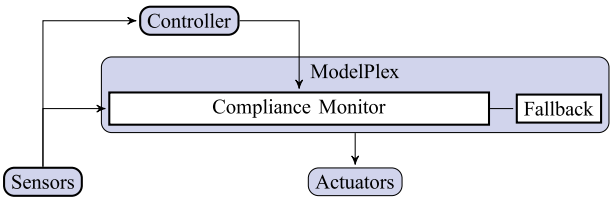


Fig. 1 ModelPlex monitors in a Simplex [39] setting: a fallback action gets executed when sensor readings and control decisions do not comply with a monitor

level challenges w.r.t. monitor implementation and violation cause diagnosis are discussed elsewhere [8,21,45].

Checking whether a system execution fits to a verified model includes checking that the actions chosen by the (unverified) controller implementation fit to *one of* the choices and requirements that the verified controller model allows. It also includes checking that the observed states can be explained by the plant model. The crucial questions are: What are the right conditions to monitor? Which monitor conditions guarantee safety without being overly restrictive? How can the correctness of such executable monitor conditions be proved formally? How can a compliance monitor be synthesized that provably represents all important aspects of complying with the verified model correctly? How much safety margin does a system need to ensure that fail-safe actions are always initiated early enough for the system to remain safe, even if its behavior ceases to comply with the model?

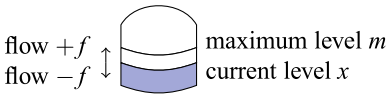
The last question is related to feedback control and can only be answered when assuming some constraints on the maximum deviation of the real system dynamics from the plant model [36]. Otherwise, i. e., if the real system might be infinitely far off from the model, safety guarantees are impossible. By the sampling theorem in signal processing [40], such constraints further enable compliance monitoring solely on the basis of sample points instead of the unobservable intermediate states about which no sensor data exists.[2]

Extension In addition to providing proofs for the results, this article extends the short version [24] with support for a *correct-by-construction* approach to synthesize ModelPlex monitors by a systematic transformation in the differential dynamic logic axiomatization [33]. We leverage an implementation of this axiomatization in our entirely new theorem prover KeYmaera X [14] by performing the ModelPlex monitor proof construction in place, as opposed to splitting it over the branches of its classical sequent calculus [29]. Sequent calculi are usually preferred for proving properties, because they induce a sequent normal form that simplifies proof construction by narrowing proof search to proof rules for top-level operators and splitting the proof over independent branches as needed. Proofs cannot close during the ModelPlex monitor construction, however, because the proof represents the conditions on system executions that the verified model imposes. That is why proof branching in our previous ModelPlex implementation [24] led to sizeable monitors with nontrivial redundancy which were simplified with (unverified) external optimization tools and, thus, had to be reverified for correctness.

Our new ModelPlex monitor synthesis presented here exploits the flexibility of differential dynamic logic axioms [33] more liberally to significantly improve locality of the construction, which leads to reductions of the resulting monitors compared to our previous approach

[2] When such constraints are not available, our method still generates verifiably correct *runtime tests*, which detect deviation from the model at the sampling points, just not between them. A fail-safe action will then lead to earliest possible best-effort mitigation of safety risks (rather than guaranteed safety).

Fig. 2 Water tank model



[24]. The axiomatic ModelPlex construction also preserves the structure in the model better. The ModelPlex construction now remains entirely under the auspices of the theorem prover without external simplification, thereby eliminating the need to reverify correctness of the resulting monitor. Efficiency during the ModelPlex monitor construction in the prover is retained using contextual rewriting in the uniform substitution calculus for differential dynamic logic [35]. We now also implemented optimizations of the ModelPlex monitor constructions as proof tactics that were previously performed manually. This leads to a fully automatic synthesis procedure for correct-by-construction ModelPlex monitors that produces proofs of correctness for the monitors it synthesizes.

2 Differential dynamic logic by example

This section recalls *differential dynamic logic* $\mathsf{d}\mathscr{L}$ [29,31,33], which we use to syntactically characterize the semantic conditions required for correctness of the ModelPlex approach. Its proof calculus [29,31,33,35] is also exploited to guarantee correctness of the specific ModelPlex monitors produced for concrete CPS models. A tactic for the proof calculus implements the correct-by-construction ModelPlex monitor synthesis algorithm.

This section also introduces a simple water tank that will be used as a running example to illustrate the concepts throughout (Fig. 2).

The water level in the tank is controlled by a digital controller that can periodically adjust flow into and from the tank by adjusting two valves. Every time the controller decides on adjusting the flow, it measures the water level through a sensor (i. e., it samples the water level). As a safety condition, we want the water tank to never overflow: any control decision of the controller must be such that the water level stays within 0 and a maximum water level m at all times. We will use this example to introduce $\mathsf{d}\mathscr{L}$ and its syntax for modeling hybrid programs step by step. The final example is repeated in Appendix 1 for easy reference.

2.1 Syntax and informal semantics

Differential dynamic logic has a notation for modeling hybrid systems as *hybrid programs*. Table 1 summarizes the relevant syntax fragment of hybrid programs together with an informal semantics. The formal semantics $\rho(\alpha)$ of hybrid program α is a relation on initial and final states of running α (recalled in Sect. 2.2 below).

Syntax of hybrid programs by example Let us start by modeling the controller of the water tank example, which can adjust two valves by either opening them or closing them.

$$(v_{\text{in}} := 1 \cup v_{\text{in}} := 0); \quad (v_{\text{out}} := 1 \cup v_{\text{out}} := 0)$$

Here, we use (deterministic) assignment $x := \theta$ to assign values to valves: setting a valve to 1, as in $v_{\text{in}} := 1$ means that the valve is open, while setting it to 0 means that the valve is closed. Now any valve can either be opened or closed, not both at the same time, which we indicate using the nondeterministic choice $\alpha \cup \beta$, as in $v_{\text{in}} := 1 \cup v_{\text{in}} := 0$. The controller first

Table 1 Hybrid program representations of hybrid systems

Statement	Effect
$\alpha; \beta$	Sequential composition, first run hybrid program α, then hybrid program β
$\alpha \cup \beta$	Nondeterministic choice, following either hybrid program α or β
α^*	Nondeterministic repetition, repeats hybrid program α $n \geq 0$ times
$x := \theta$	Assign value of term θ to variable x (discrete jump)
$x := *$	Assign arbitrary real number to variable x
$?F$	Check that a particular condition F holds, and abort if it does not
$(x_1' = \theta_1, \ldots,$	Evolve x_i along differential equation system $x_i' = \theta_i$
$x_n' = \theta_n \ \& \ F)$	Restricted to maximum evolution domain F

adjusts the incoming valve v_{in}, before it adjusts the outgoing valve v_{out}, as modeled using the sequential composition $\alpha; \ \beta$.

For theorem proving, however, it often makes sense to describe the system at a more abstract level in order to keep the model simple. Let us, therefore, replace the two valves with their intended effect of adjusting water flow f.

$$f := *; \ ?(-1 \leq f \leq 1)$$

Here, we use nondeterministic assignment $f := *$, which assigns an arbitrary real number to f, so we abstractly model that the controller will somehow choose water flow. Next, we need to restrict this arbitrary flow to those flows that make sense. Let us assume that the incoming and the outgoing pipe from our water tank can provide and drain at most 1 liter per second, respectively. For this, we use the test $?(-1 \leq f \leq 1)$, which checks that $-1 \leq f \leq 1$ holds, and aborts the execution attempt if it does not. Together, the nondeterministic assignment and test mean that the controller can choose any flow in the interval $f \in [-1, 1]$.

Now that we know the actions of the controller, let us add the physical response, often called *plant*, using differential equations. We use x to denote the current water level in the water tank.

$$f := *; \ ?(-1 \leq f \leq 1); \ \{x' = f \ \& \ x \geq 0\}$$

The idealized differential equation $x' = f$ means that the water level evolves according to the chosen flow. This considerably simplifies water flow models (e. g., it neglects the influence of water level on flow, and flow disturbance in pipes). The evolution domain constraint $x \geq 0$ models a physical constraint that the water level can never be less than empty. Otherwise, the differential equation would include negative water content in the tank below zero on negative flow, because differential equations evolve for an arbitrary amount of time (even for time 0), as long as their evolution domain constraint is satisfied. Note, that when the tank is empty ($x = 0$) and the controller still chooses a negative flow $f < 0$ as permitted by the test $?(-1 \leq f \leq 1)$, the evolution domain constraint $x \geq 0$ in the ODE will abort immediately. As a result, only non-negative values for f will make progress in case the tank is empty. This model means that the controller can choose flow exactly once, and then the water level evolves according to that flow for some time. Next, we include a loop, indicated by the Kleene star, so that the controller and the plant can run arbitrarily many times.

$$\big(f := *;\ ?(-1 \le f \le 1);\ \{x' = f\ \&\ x \ge 0\}\big)^*$$

However, this model provides no guarantees whatsoever on the time that will pass between two controller executions, since differential equations are allowed to evolve for an arbitrary amount of time. In order to guarantee that the controller runs at least every ε time, we model controller periodicity and sampling period by adding the differential equation $t' = 1$ to capture time, and a constraint $t \le \varepsilon$ to indicate that at most ε time can pass until the plant must stop executing and hand over to the controller again. We reset the stopwatch t after each controller run using $t := 0$.

$$\big(f := *;\ ?(-1 \le f \le 1);\ t := 0;\ \{x' = f,\ t' = 1\ \&\ x \ge 0 \wedge t \le \varepsilon\}\big)^*$$

Note, that through $t \le \varepsilon$ the sampling period does not need to be the same on every control cycle, nor does it need to be exactly ε time.

Now that we know the sampling period, let us make one final adjustment to the controller: It actually cannot always be safe to choose positive inflow, as allowed by the test $?-1 \le f \le 1$ (e. g., it would be unsafe if the current water level x is already at the maximum m). Since we know that the controller will run again at the latest in ε time, we can choose inflow such that it will not exceed the maximum level m until then, as summarized below.

$$\left(f := *;\ ?\left(-1 \le f \le \frac{m-x}{\varepsilon}\right);\ t := 0;\ \{x' = f,\ t' = 1\ \&\ x \ge 0 \wedge t \le \varepsilon\}\right)^*$$

Differential dynamic logic syntax by example Next, we want to prove that this program is correct. For this, we first need to find a formal safety condition that captures correctness. Since we want the tank to never overflow, all runs of the program must ensure $0 \le x \le m$, which in $\mathsf{d}\mathscr{L}$ is expressed using the *box modality* $[\alpha]\phi$. The formula $[\alpha]\phi$ is not true in all initial states, only in those that at least satisfy $0 \le x <\le m$ to begin with. The modeling idiom $\phi \to [\alpha]\psi$ expresses that, when started in an initial state that satisfies the initial condition ϕ, then all runs of the model α result in states that satisfy ψ, similar to a Hoare triple. Formula (1) below summarizes the water tank model and the safety condition using this idiom.

$$\underbrace{0 \le x \le m \wedge \varepsilon > 0}_{\phi} \to \Big[\Big(f := *;\ ?\left(-1 \le f \le \frac{m-x}{\varepsilon}\right);\ t := 0;\ \{x' = f,\ t' = 1\ \&\ x \ge 0 \wedge t \le \varepsilon\}\Big)^*\Big]\overbrace{(0 \le x \le m)}^{\psi} \tag{1}$$

This formula expresses that, when started with a safe water level between 0 and maximum ($0 \le x \le m$) and with some positive sampling period ($\varepsilon > 0$), our water tank model will keep the water level between 0 and maximum. It is provable in the $\mathsf{d}\mathscr{L}$ proof calculus.

Syntax summary Sequential composition $\alpha; \beta$ says that β starts after α finishes. The nondeterministic choice $\alpha \cup \beta$ follows either α or β. The nondeterministic repetition operator α^* repeats α zero or more times. Assignment $x := \theta$ instantaneously assigns the value of term θ to the variable x, while $x := *$ assigns an arbitrary value to x. The test $?F$ checks that a condition F holds, and aborts if it does not. $x' = \theta\ \&\ F$ describes a continuous evolution of x within the evolution domain F.

The set of $\mathsf{d}\mathscr{L}$ formulas is generated by the following grammar ($\sim \in \{<, \le, =, \ge, >\}$ and θ_1, θ_2 are arithmetic expressions in $+, -, \cdot, /$ over the reals):

$$\phi ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \to \psi \mid \forall x \phi \mid \exists x \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

$\mathsf{d}\mathscr{L}$ allows us to make statements that we want to be true for all runs of a hybrid program ($[\alpha]\phi$) or for at least one run ($\langle\alpha\rangle\phi$). Both constructs are necessary to derive safe monitors: we need $[\alpha]\phi$ proofs so that we can be sure all behavior of a model are safe; we need $\langle\alpha\rangle\phi$ proofs to find monitor specifications that detect whether or not a system execution fits to the verified model. Differential dynamic logic comes with a verification technique to prove correctness properties of hybrid programs (cf. [33] for an overview of $\mathsf{d}\mathscr{L}$ and KeYmaera, and [14] for an overview of KeYmaera X).

2.2 Formal semantics of $\mathsf{d}\mathscr{L}$

ModelPlex is based on a transition semantics instead of trace semantics [31], since it is easier to handle and fits to checking monitors at sample points.

The semantics of $\mathsf{d}\mathscr{L}$, as defined in [29], is a Kripke semantics in which states of the Kripke model are states of the hybrid system. Let $\mathbb{R}$ denote the set of real numbers, and V denote the set of variables. A state is a map $\nu : V \to \mathbb{R}$; the set of all states is denoted by Sta. We write $\nu \models \phi$ if formula ϕ is true at state ν (Definition 2). Likewise, $[\![\theta]\!]_\nu$ denotes the real value of term θ at state ν, while $\nu(x)$ denotes the real value of variable x at state ν. The semantics of HP α is captured by the state transitions that are possible by running α. For continuous evolutions, the transition relation holds for pairs of states that can be interconnected by a continuous flow respecting the differential equation and invariant region. That is, there is a continuous transition along $x' = \theta \,\&\, H$ from state ν to state ω, if there is a solution of the differential equation $x' = \theta$ that starts in state ν and ends in ω and that always remains within the region H during its evolution.

Definition 1 (Transition semantics of hybrid programs) The *transition relation* ρ specifies which state ω is reachable from a state ν by operations of α. It is defined as follows.

1. $(\nu, \omega) \in \rho(x := \theta)$ iff $\omega(x) = [\![\theta]\!]_\nu$ and $\nu(z) = \omega(z)$ for all state variables $z \neq x$.
2. $(\nu, \omega) \in \rho(x := *)$ iff $\nu(z) = \omega(z)$ for all state variables $z \neq x$.
3. $(\nu, \omega) \in \rho(?\phi)$ iff $\nu = \omega$ and $\nu \models \phi$.
4. $(\nu, \omega) \in \rho(x_1' = \theta_1, \ldots, x_n' = \theta_n \,\&\, H)$ iff for some $r \geq 0$, there is a (*flow*) function $\varphi{:}[0, r] \to \text{Sta}$ with $\varphi(0) = \nu$, $\varphi(r) = \omega$, such that for each time $\zeta \in [0, r]$ the differential equation holds and the evolution domain is respected $\varphi(\zeta) \models x_i' = \theta_i \& H$, see [29,34] for details
5. $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
6. $\rho(\alpha; \beta) = \{(\nu, \omega) : (\nu, \mu) \in \rho(\alpha), (\mu, \omega) \in \rho(\beta) \text{ for a state } \mu\}$
7. $\rho(\alpha^*) = \bigcup_{n \in \mathbb{N}} \rho(\alpha^n)$ where $\alpha^{i+1} \hat{=} (\alpha; \alpha^i)$ and $\alpha^0 \hat{=} ?true$.

Definition 2 (Interpretation of $\mathsf{d}\mathscr{L}$ formulas) The interpretation $\models$ of a $\mathsf{d}\mathscr{L}$ formula with respect to state ν is defined as follows.

1. $\nu \models \theta_1 \sim \theta_2$ iff $[\![\theta_1]\!]_\nu \sim [\![\theta_2]\!]_\nu$ for $\sim \in \{=, \leq, <, \geq, >\}$
2. $\nu \models \phi \wedge \psi$ iff $\nu \models \phi$ and $\nu \models \psi$, accordingly for $\neg, \vee, \rightarrow, \leftrightarrow$
3. $\nu \models \forall x\, \phi$ iff $\omega \models \phi$ for all ω that agree with ν except for the value of x
4. $\nu \models \exists x\, \phi$ iff $\omega \models \phi$ for some ω that agrees with ν except for the value of x
5. $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all ω with $(\nu, \omega) \in \rho(\alpha)$
6. $\nu \models \langle\alpha\rangle\phi$ iff $\omega \models \phi$ for some ω with $(\nu, \omega) \in \rho(\alpha)$

We write $\models \phi$ to denote that ϕ is *valid*, i. e., that $\nu \models \phi$ for all states ν.

2.3 Notation and supporting lemmas

$\mathrm{BV}(\alpha)$ denotes the bound variables [35] in α, i.e., those written to in α, $\mathrm{FV}(\psi)$ are free variables [35] in ψ, Σ is the set of all variables, and $A \setminus B$ denotes the set of variables being in some set A but not in some other set B. Furthermore, $\nu|_A$ denotes the state ν projected to just the variables in A, whereas ν_x^y denotes the state ν in which x is interpreted as y.

In the proofs throughout this article, we will use the following lemmas specialized from [35, Lemmas 12, 14, and 15]. Hybrid programs only change their bound variables:

Lemma 1 *(Bound effect lemma) If $(\nu, \omega) \in \rho(\alpha)$, then $\nu = \omega$ on $\Sigma \setminus BV(\alpha)$.*

The truth of formulas only depends on their free variables:

Lemma 2 *(Coincidence lemma) If $\nu = \tilde{\nu}$ on $FV(\phi)$ then $\nu \models \phi$ iff $\tilde{\nu} \models \phi$.*

Similar states (that agree on the free variables) have similar transitions:

Lemma 3 *(Coincidence lemma) If $\nu = \tilde{\nu}$ on $V \supseteq FV(\alpha)$ and $(\nu, \omega) \in \rho(\alpha)$, then there is an $\tilde{\omega}$ such that $(\tilde{\nu}, \tilde{\omega}) \in \rho(\alpha)$ and $\omega = \tilde{\omega}$ on V.*

The notation $\nu|_V = \tilde{\nu}|_V$ is used interchangeably with $\nu = \tilde{\nu}$ agree on V.

3 ModelPlex approach for verified runtime validation

CPS are almost impossible to get right without sufficient attention to prior analysis, for instance by formal verification and formal validation techniques. We assume to be given a verified model of a CPS, i.e. formula (2) is proved valid,[3] for example using the differential dynamic logic proof calculus [29,33] implemented in KeYmaera [37] and KeYmaera X [14]:

$$\phi \rightarrow [\alpha^*]\psi \tag{2}$$

Formula (2) expresses that all runs of the hybrid system α^*, which start in states that satisfy the precondition ϕ and repeat α arbitrarily many times, only end in states that satisfy the postcondition ψ. Note, that in this article we discuss models of the form α^* for comprehensibility reasons. The approach is also applicable to more general forms of models (e.g., models without loops, or models where only parts are executed in loops).

The model α^* is a *hybrid system model* of a CPS, which means that it describes both the discrete control actions of the controllers in the system and the continuous physics of the plant and the system's environment. For example, our running example of a water tank repeated below models a hybrid system, which consists of a controller that chooses flow and a plant that determines how the water level changes depending on the chosen flow.

$$\alpha^* \equiv \left(f := *;\ ?\left(-1 \leq f \leq \frac{m-x}{\varepsilon}\right);\ t := 0;\ \{x' = f,\ t' = 1 \,\&\, x \geq 0 \wedge t \leq \varepsilon\} \right)^*$$

Formula (2) is proved using some form of induction with *invariant* φ, i.e., a formula for which the following three formulas are provable:

$$\varphi \text{ is an invariant for (2), i.e., } \quad \varphi \rightarrow [\alpha]\varphi \qquad \phi \rightarrow \varphi \qquad \varphi \rightarrow \psi \tag{3}$$

[3] We use differential dynamic logic ($\mathsf{d}\mathscr{L}$) and KeYmaera and KeYmaera X as a theorem prover to illustrate our concepts throughout this article. The concept of ModelPlex is not predicated on the use of KeYmaera/KeYmaera X to prove (2). Other verification techniques could be used to establish validity of this formula. The flexibility of the underlying logic $\mathsf{d}\mathscr{L}$, its support for both $[\alpha]\phi$ and $\langle\alpha\rangle\phi$, and its proof calculus, however, are exploited for systematically constructing monitors from proofs in the rest of the article.

which shows that a loop invariant φ holds after every run of α if it was true before (i.e., $\varphi \rightarrow [\alpha]\varphi$), that the loop invariant holds initially ($\phi \rightarrow \varphi$) and implies the postcondition ($\varphi \rightarrow \psi$).

However, since we usually made approximations when modeling the controller and the physics, and since failures and other deviations may occur in reality (e.g., a valve could fail), we cannot simply transfer this safety proof to the real system. The safety guarantees that we obtain by proving formula (2) about the model α^* transfer to the real system, *if* the actual CPS execution fits to α^*.

Example 1 (What to monitor) Let us recall the water tank example. First, since failures may occur we need to monitor actual evolution, such as that the actual water level corresponds to the level expected by the chosen valve positions and the actual time passed between controller executions does not exceed the modeled sampling period. The monitor needs to allow some slack around the expected water level to compensate for the neglected physical phenomena. Sections 3.2 and 3.5 describe how to synthesize such model monitors automatically. Second, the controller implementation differs from the model, e.g., it might follow different filling strategies, so we need to check that the implemented controller only chooses flows f that satisfy $-1 \leq f \leq \frac{m-x}{\varepsilon}$. Section 3.4 describes how to synthesize such controller monitors automatically. Finally, we can monitor controller decisions for the expected real-world effect, since the hybrid system model contains a model of the physics of the water tank. Section 3.6 describes how to synthesize such prediction monitors automatically. The controller in the model, which is verified to be safe, gives us a fail-safe action that we can execute instead of the unverified controller implementation when one of the monitors is not satisfied.

Since we want to preserve safety properties, a CPS γ fits to a model α^*, *if* the CPS reaches at most those states that are reachable by the model, i.e., $\rho(\gamma) \subseteq \rho(\alpha^*)$ [27], because all states reachable by α^* from states satisfying ϕ are safe by (2). For example, a controller that chooses inflow more cautiously, such as only half the maximum inflow from the model, i.e., $f \leq \frac{m-x}{2\varepsilon}$, would also be safe. So would be running the controller more frequently than every ε time, but not less frequently.

However, we do not know the true CPS γ precisely,[4] so we cannot use refinement-based techniques (e.g., [27]) to prove that the true CPS γ refines the model α^*. Therefore, we need to find a condition based on α^* that we can check at runtime to see if concrete runs of the true CPS γ behave like the model α^*.

Example 2 (Canonical monitor candidates) A monitor condition that would be easy to check is to monitor the postcondition ψ (e.g., monitor the safety condition of the water tank $0 \leq x \leq m$). But that monitor is unsafe, because if ψ is violated at runtime, the system is already unsafe and it is too late to do anything about it (e.g., the water tank did already overflow). Another monitor that would be easy to check is the invariant φ used to prove Formula (2). But that monitor is also unsafe, because once φ is violated at runtime, the controller is no longer guaranteed to be safe, since Formula (3) only proves it to be safe when maintaining invariant φ (e.g., in the water tank example, the invariant $\varphi \equiv 0 \leq x \leq m$ is not even stronger than the safety condition). But if we detect when a CPS is about to deviate from α *before* leaving φ, we can still switch to a fail-safe controller to avoid $\neg\psi$ from ever happening (see Fig. 3). Yet even so, the invariant φ will not even contain all conditions that need to be monitored, since φ only reflects what will not change when running the particular model α, which says nothing about the behavior of the true CPS γ.

[4] It is an annoying fact of physics that there will never quite be a perfect model of γ.

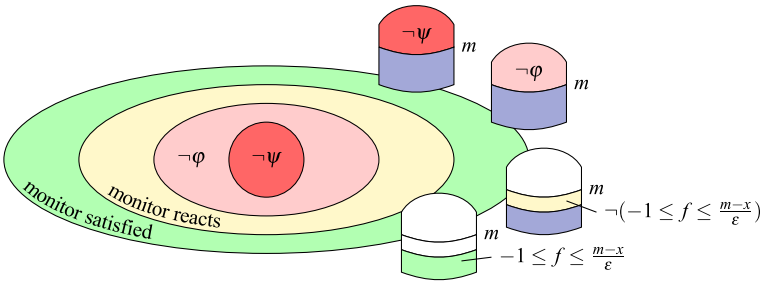


Fig. 3 States when safety measures are required according to postcondition ψ, invariant φ, and monitor. The water tanks illustrate water levels corresponding to these conditions

The basic idea behind ModelPlex is based on online monitoring: we periodically sample γ to obtain actual system states ν_i. A state ν_i includes values for each of the bound variables (i. e., those that are written) from the model α^*. For example, for our water tank we need to sample flow f (written to in $f := *$), water level x (written to in $x' = f$), and time t (written to in $t := 0$ and $t' = 1$). We then check pairs of such states for being included in the reachability relation of the model, which is expressed in $\mathsf{d}\mathscr{L}$ semantics as $(\nu_{i-1}, \nu_i) \in \rho(\alpha^*)$. We will refer to the first state in such a pair by *prior* state and to the second one by *posterior* state. This is the right semantic condition to check, but not computationally represented. The important question answered by ModelPlex through automatic synthesis is how that check can be represented in a monitor condition in an easily and efficiently computable form.

Example 3 (Desired arithmetic monitor representation) For example, by manually analyzing the hybrid program of the water tank example, the result is expected to be the following real arithmetic formula. The annotations under the braces refer to the part of the hybrid program of the water tank that points us to the corresponding condition.

$$\underbrace{-1 \leq \nu_i(f) \leq \frac{m - \nu_{i-1}(x)}{\varepsilon}}_{\text{(i) } f := *;\ ?\left(-1 \leq f \leq \frac{m-x}{\varepsilon}\right)} \wedge \underbrace{\nu_i(x) = \nu_{i-1}(x) + \nu_i(f)\nu_i(t)}_{\text{(ii) } x'=f,\ t'=1} \wedge \underbrace{\nu_{i-1}(x) \geq 0 \wedge \nu_i(x) \geq 0 \wedge 0 \leq \nu_i(t) \leq \varepsilon}_{\text{(iii) } t := 0,\ \&\, x \geq 0 \wedge t \leq \varepsilon}$$

This formula describes that (i) the flow $\nu_i(f)$ in the posterior state has to obey certain bounds, depending on the prior water level $\nu_{i-1}(x)$, resulting from the nondeterministic assignment and the test; (ii) the posterior water level $\nu_i(x)$ is given by the solution of the differential equation $x + \int f dt = x + ft$, i. e., the posterior water level should be equal to the prior water level $\nu_{i-1}(x)$ plus the amount resulting from flow $\nu_i(f)$ in time $\nu_i(t)$; finally, (iii) the evolution domain constraints must be true, meaning the posterior water level must be non-negative and the time $\nu_i(t)$ must be between 0 and ε. Note, that it is tempting to just read off a wrong condition $\nu_i(t) = 0$ from hybrid program $t := 0$. Since t is not constant in the ODE following the assignment ($t := 0;\ t' = 1$), this condition must be phrased $0 \leq \nu_i(t)$. Also note, that it is very easy to get the evolution domain wrong: evolution domain constraints have to hold throughout the ODE, which includes the beginning and the end, so the check must include both $\nu_{i-1}(x) \geq 0$ and $\nu_i(x) \geq 0$. The sound proof calculus of $\mathsf{d}\mathscr{L}$ prevents such mistakes when deriving monitor conditions.

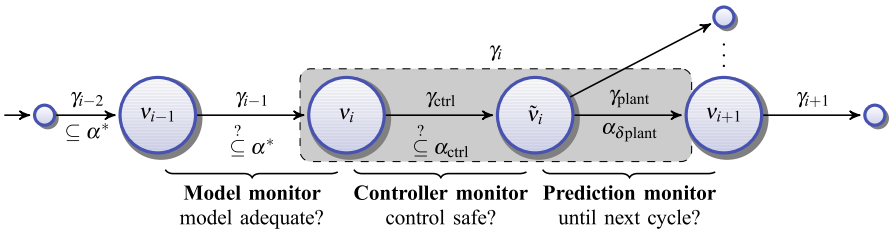


Fig. 4 Use of ModelPlex monitors along a system execution

The question is: How to find such an arithmetic representation automatically from just the formula (1)? And how to prove its correctness? ModelPlex derives three kinds of such formulas as monitors (model monitor, controller monitor, and prediction monitor, cf. Fig. 4) that check the behavior of the actual CPS at runtime for compliance with its model. These monitors have the following characteristics.

Model monitor The model monitor checks the previous state ν_{i-1} and current state ν_i for compliance with the model, i. e.. whether the observed transition from ν_{i-1} to ν_i is compatible with the model. In each state ν_i we test the sample point ν_{i-1} from the previous execution γ_{i-1} for deviation from α^*, i. e., test $(\nu_{i-1}, \nu_i) \in \rho(\alpha^*)$. If violated, other verified properties may no longer hold for the system so a failsafe action is initiated. The system itself, however, still satisfies safety condition ψ if the prediction monitor was satisfied at ν_{i-1}. Frequent violations indicate an inadequate model that should be revised to better reflect reality.

Controller monitor The controller monitor checks the output of a controller implementation against the correct controller model. If the controller implementation performs an action that the controller model allows in the present state, then it has been verified offline to be safe by Formula (2). Otherwise, the action is discarded and replaced by a default action that has been proved safe. In intermediate state $\tilde{\nu}_i$ we test the current controller decisions of the controller implementation γ_{ctrl} for compliance with the model, i. e., test $(\nu_i, \tilde{\nu}_i) \in \rho(\alpha_{\text{ctrl}})$. The controller α_{ctrl} will be obtained from the model α^* through proof steps. Controller monitors have some similarities with Simplex [39], which is designed for switching between verified and unverified controllers. The controller monitor, instead, corresponds to the more general idea of testing contracts dynamically at runtime while defaulting to a specified default action choice if the contract fails. If a controller monitor is violated, commands from a fail-safe controller replace the current controller's decisions to ensure that no unsafe commands are ever actuated.

Prediction monitor The model monitor detects deviations from the model as soon as possible on the measured data, but that may already have made the system unsafe. The role of the prediction monitor is to check the impact of bounded deviations from the model to predict whether the next state could possibly become unsafe upon deviation from the model so that a corrective action is advised. If the actual execution stays far enough away from unsafe states, the prediction monitor will not intervene because no disturbance within the bound could make it unsafe. In intermediate state $\tilde{\nu}_i$ we test the safety impact of the *current controller decision* w.r.t. the predictions of a bounded deviation plant model $\alpha_{\delta\text{plant}}$, which has a tolerance around the model plant α_{plant}, i. e., check $\nu_{i+1} \models \varphi$ for all ν_{i+1} such that $(\tilde{\nu}_i, \nu_{i+1}) \in \rho(\alpha_{\delta\text{plant}})$. Note, that we simultaneously check all ν_{i+1} by checking a characterizing condition of $\alpha_{\delta\text{plant}}$ at $\tilde{\nu}_i$. If violated, the current control choice

is not guaranteed to keep the system safe under all disturbances until the next control cycle and, thus, a fail-safe controller takes over.

A simulation illustrating the effect of these monitors on the water tank running example will be discussed in Fig. 11, where an unsafe controller and small deviation from the idealistic model would result in violation of the safety property, if not corrected by the monitors synthesized in this article.

The assumption for the prediction monitor is that the real execution is not arbitrarily far off the plant models used for safety verification, because otherwise safety guarantees can be neither made on unobservable intermediate states nor on safety of the future system evolution [36]. We propose separation of disturbance causes in the models: ideal plant models α_{plant} for correctness verification purposes, implementation deviation plant models $\alpha_{\delta\text{plant}}$ for monitoring purposes. We support any deviation model (e. g., piecewise constant disturbance, differential inclusion models of disturbance), as long as the deviation is bounded and differential invariants can be found. We further assume that monitor evaluations are at most some ε time units apart (e. g., along with a recurring controller execution). Note that disturbance in $\alpha_{\delta\text{plant}}$ is more manageable compared to a model of the form α^*, because we can focus on single runs α instead of repetitions for guaranteed monitoring purposes.

3.1 Characterizing semantic relations between states in logic

All ModelPlex monitors relate states, albeit for different purposes to safeguard different parts of the CPS execution (Fig. 4). States are semantic objects and as such cannot be related, manipulated, or even just represented precisely in a program. This section develops a systematic logical characterization as syntactic expressions for such state relations, which will ultimately lead to computable programs for the corresponding monitor conditions. We systematically derive a check that inspects states of the actual CPS to detect deviation from the model α. We first establish a notion of state recall and show that compliance of an execution from state ν to ω with α can be characterized syntactically in dL.

The ModelPlex monitoring principle illustrated in Fig. 4 is intuitive, but its sequence of states ν_i is inherently semantic and, thus, inaccessible in syntactic programs. Our first step is to introduce a vector of logical variables x and x^+ for the symbolic prior and posterior state variables. The basic idea is that ModelPlex monitors identify conditions on the relationships between the values of prior and posterior state expressed as a logical formula involving the variables x and x^+. Concrete states ν_{i-1} and ν_i can then be fed into the monitor formula as the real values for the variables x and x^+ to check whether the monitor is satisfied along the actual system execution.

Definition 3 and Lemma 4 below describe central ingredients for online monitoring in this article and are true for models β of arbitrary form (not just for models α^* with a loop).

Definition 3 (State recall) Let V denote the set of variables whose state we want to recall. We use the formula $\Upsilon^+ \equiv \bigwedge_{x\in V} x = x^+$ to express a characterization of the values of variables x in a state posterior to a run of β, where we always assume the fresh variables x^+ to occur solely in Υ^+. The variables in x^+ can be used to recall this state. We define the satisfaction relation $(\nu, \omega) \models \phi$ of dL formula ϕ for a pair of states (ν, ω) as ϕ evaluated in the state resulting from ν by interpreting x^+ as $\omega(x)$ for all $x \in V$, i. e., $(\nu, \omega) \models \phi$ iff $\nu_{x^+}^{\omega(x)} \models \phi$.

This enables a key ingredient for ModelPlex: establishing a direct correspondence of a semantic reachability of states with a syntactic logical formula internalizing that semantic relationship by exploiting the $\langle\cdot\rangle$ modality of dL.

Lemma 4 *(Logical state relation) Let $V = BV(\beta)$. Two states ν, ω that agree on $\Sigma \setminus V$, i.e., $\nu|_{\Sigma\setminus V} = \omega|_{\Sigma\setminus V}$, i.e., $\nu(z) = \omega(z)$ for all $z \in \Sigma \setminus V$, satisfy $(\nu, \omega) \in \rho(\beta)$ iff $(\nu, \omega) \models \langle\beta\rangle\Upsilon^+$.*

Proof "$\Rightarrow$" Let $(\nu, \omega) \in \rho(\beta)$. Since ν and $\nu_{x^+}^{\omega(x)}$ agree except on x^+, which are not free variables of β, $(\nu, \omega) \in \rho(\beta)$ also implies by coincidence Lemma 3 that there is a $\tilde{\omega}$ such that $(\nu_{x^+}^{\omega(x)}, \tilde{\omega}) \in \rho(\beta)$ and $\omega = \tilde{\omega}$ except on x^+. Now $(\nu_{x^+}^{\omega(x)}, \tilde{\omega}) \in \rho(\beta)$ implies that $\nu_{x^+}^{\omega(x)} = \tilde{\omega}$ agree except on $BV(\beta)$ by bound effect Lemma 1. Hence, $\nu_{x^+}^{\omega(x)} = \tilde{\omega}$ agree on x^+ since $x^+ \notin BV(\beta)$ and, thus, also $\omega_{x^+}^{\omega(x)} = \tilde{\omega}$ on x^+. Since $\omega = \tilde{\omega}$ agree except on x^+ and $\omega_{x^+}^{\omega(x)} = \tilde{\omega}$ agree on x^+, also $\omega_{x^+}^{\omega(x)} = \tilde{\omega}$ agree everywhere, which implies, $(\nu_{x^+}^{\omega(x)}, \omega_{x^+}^{\omega(x)}) \in \rho(\beta)$, because $(\nu_{x^+}^{\omega(x)}, \tilde{\omega}) \in \rho(\beta)$. As $\omega_{x^+}^{\omega(x)} \models x = x^+$ for all x, so $\omega_{x^+}^{\omega(x)} \models \Upsilon^+$. Consequently, $\nu_{x^+}^{\omega(x)} \models \langle\beta\rangle\Upsilon^+$, which is $(\nu, \omega) \models \langle\beta\rangle\Upsilon^+$.

"$\Leftarrow$" Let $(\nu, \omega) \models \langle\beta\rangle\Upsilon^+$, that is, $\nu_{x^+}^{\omega(x)} \models \langle\beta\rangle\Upsilon^+$. So there is a $\tilde{\omega}$ such that $(\nu_{x^+}^{\omega(x)}, \tilde{\omega}) \in \rho(\beta)$ and $\tilde{\omega} \models \Upsilon^+$. Now $\tilde{\omega} \models \Upsilon^+$ implies that $\tilde{\omega}(x) = \tilde{\omega}(x^+)$. By the bound effect Lemma 1, $\nu_{x^+}^{\omega(x)} = \tilde{\omega}$ agree except on $BV(\beta)$. Thus, $\tilde{\omega}(x^+) = \nu_{x^+}^{\omega(x)}(x^+) = \omega(x)$ for all $x \in BV(\beta)$ as $x^+ \notin BV(\beta)$. Combining both yields that $\tilde{\omega} = \omega$ agree on all $x \in BV(\beta)$. Since $(\nu_{x^+}^{\omega(x)}, \tilde{\omega}) \in \rho(\beta)$ and $\nu_{x^+}^{\omega(x)} = \nu$ agree except on $x^+ \notin FV(\beta)$, coincidence Lemma 3 implies there is a μ such that $(\nu, \mu) \in \rho(\beta)$ and $\mu = \tilde{\omega}$ agree except on x^+. So, $\mu = \tilde{\omega} = \omega$ agree on $x \in BV(\beta)$. And $\mu = \nu$ agree except on $x \in BV(\beta)$ by bound effect Lemma 1. From the assumption that $\nu = \omega$ agree except on $BV(\beta)$, it follows that $\mu = \omega$ also on $\Sigma \setminus BV(\beta)$, so $\mu = \omega$. Hence, $(\nu, \mu) \in \rho(\beta)$ implies $(\nu, \omega) \in \rho(\beta)$. □

Suppose the CPS executed for some period of time and made it from state ν to a state ω. That transition fits to the verified model α^* iff the semantic condition $(\nu, \omega) \in \rho(\alpha^*)$ holds, i. e., the states ν, ω are in the transition relation induced by the semantics of α^*. The syntactic formula $\langle\alpha^*\rangle\Upsilon^+$ expresses something like that. Lemma 4 enables us to use formula (4) as a starting point to find compliance checks systematically.

$$\langle\alpha^*\rangle\Upsilon^+ \tag{4}$$

The logical formula (4) relates a prior state of a CPS to its posterior consecutive state through at least one path through the model α^*.[5] The formula (4) is satisfied in a state ν, if there is at least one run of the model α^* starting in the state ν and resulting in a state ω recalled using Υ^+. In other words, at least one path through α^* explains how the prior state ν got transformed into the posterior state ω.

In principle, formula (4) would already be a perfect monitor for the question whether the state change to Υ^+ can be explained by model α^*. But formula (4) is hard if not impossible to evaluate at runtime efficiently, because it refers to a hybrid system α^*, which includes loops, nondeterminism, and differential equations and is, thus, difficult to execute without nontrivial backtracking and differential equation solving. Yet, any formula that is equivalent to or implies (4) but is easier to evaluate in a state is a correct monitor as well.

To simplify formula (4), we use theorem proving to find a quantifier-free first-order real arithmetic form so that it can be evaluated efficiently at runtime. The resulting first-order real arithmetic formula can be easily implemented in a runtime monitor that is evaluated by

[5] Consecutive states for α^* mean before and after executions of α (i. e., in $\alpha;\alpha;\alpha;$ at the positions indicated with an arrow, not within α).

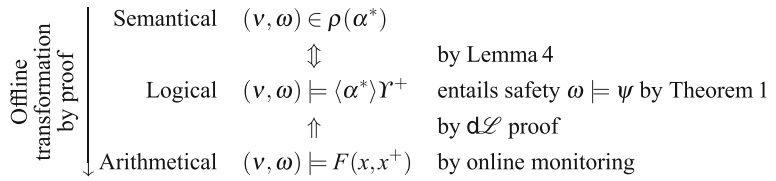


Fig. 5 Semantical representation, logic characterization, and arithmetical form of a model monitor. Monitor synthesis translates between these representations offline

plugging the concrete values in for x and x^+. A monitor is executable code that only returns true if the transition from the prior system state to the posterior state is compliant with the model. Thus, deviations from the model can be detected at runtime, so that appropriate fallback and mitigation strategies can be initiated.

3.2 Model monitor synthesis

This section introduces the nature of ModelPlex monitor specifications, which form the basis of our correct-by-construction synthesis procedure for ModelPlex monitors. Here, we focus on the ModelPlex model monitor, but its principles continue to apply for the controller and prediction monitors, as elaborated subsequently.

Figure 5 gives an overview of the offline synthesis process for model monitors. Semantically, a monitor is a check that a pair of states (ν, ω) is contained in the transition relation $\rho(\alpha^*)$ of the monitored hybrid systems model α^* (Fig. 6). This corresponds to our intuitive understanding of a monitor: through sensors, we observe states of a system, and want to know if those observations fit to the model α^* of the system. By Lemma 4, the syntactic counterpart in the logic dℒ of this semantic condition $(\nu, \omega) \in \rho(\alpha^*)$ is the logical formula $\langle\alpha^*\rangle\Upsilon^+$ from (4). The dℒ formula (4) syntactically characterizes the semantic statement that the hybrid system model α^* can reach a posterior state[6] characterized by x^+ from the prior state characterized by x. The dℒ formula (4) is a perfect logical monitor but difficult to execute quickly, so we are looking for easier logical formulas $F(x, x^+)$ that are equivalent to or imply formula (4). ModelPlex uses theorem proving to systematically synthesize a provably correct real arithmetic formula $F(x, x^+)$ in a correct-by-construction approach.[7]

The intuition is that formula (4) holds because all conditions hold that are identified as implying formula (4) in its proof. Some of these conditions hold always (subgoals that can be proved to be valid always) while others will be checked at runtime whether they hold (subgoals that do not always hold but only during executions that fit to the particular hybrid system α^*). If the ModelPlex monitor is satisfied at runtime, then the proof implying formula (4) holds *in the current CPS execution*.

Note, that computationally expensive operations, such as quantifier elimination, are *performed offline* in this process and only arithmetic evaluation for concrete state values remains to be done online. If the ModelPlex specification (4) does not hold for the variable values from a prior and posterior state during the CPS execution (checked by evaluating $F(x, x^+)$ on observations), then that behavior does not comply with the model (e. g., the wrong control action was taken under the wrong circumstances, unanticipated dynamics in the environment

[6] Recall that $\Upsilon^+ \equiv \bigwedge_{x\in V} x = x^+$ for variables V.

[7] The formula $F(x, x^+)$ implies $\langle\alpha^*\rangle\Upsilon^+$, because we will use non-equivalence proof steps to derive $F(x, x^+)$ from $\langle\alpha^*\rangle\Upsilon^+$.

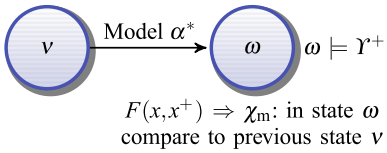


Fig. 6 A model monitor checks that two states ν and ω are contained in the transition relation of the program $(\nu, \omega) \in \rho(\alpha^*)$; the posterior state ω is captured in x^+ through Υ^+

occurred, sensor uncertainty led to unexpected values, or the system was applied outside the specified operating environment).

Intuitively, a *model monitor* χ_m is correct when the monitor entails safety if it is satisfied on consecutive observations, which is formalized in Theorem 1 below. Note, that Theorem 1 for models β without loops follows immediately from Lemma 4 and the safety proof. Thanks to Lemma 4, correctness of model monitors is also easy to prove:

Theorem 1 *(Model monitor correctness) Let α^* be provably safe, so $\models \phi \rightarrow [\alpha^*]\psi$ and let $V = BV(\alpha^*)$. Let $\nu_0, \nu_1, \nu_2, \nu_3 \ldots \in \mathbb{R}^n$ be a sequence of states that agree on $\Sigma \backslash V$, i.e., $\nu_0|_{\Sigma\backslash V} = \nu_k|_{\Sigma\backslash V}$ for all k, and that start in $\nu_0 \models \phi$. If $(\nu_i, \nu_{i+1}) \models \chi_m$ for all $i < n$, then $\nu_n \models \psi$ where*

$$\chi_m \equiv \langle \alpha^* \rangle \Upsilon^+ \tag{5}$$

Proof Show $(\nu_0, \nu_n) \in \rho(\alpha^*)$ by induction over n, such that $\models \phi \rightarrow [\alpha^*]\psi$ and $\nu_0 \models \phi$ imply $\nu_n \models \psi$. If $n = 0$ then $(\nu_0, \nu_0) \in \rho(\alpha^*)$ trivially by Definition 1. For $n + 1 > 0$ assume $(\nu_0, \nu_n) \in \rho(\alpha^*)$ and $(\nu_n, \nu_{n+1}) \models \langle \alpha^* \rangle \Upsilon^+$. By Lemma 4, $(\nu_n, \nu_{n+1}) \models \langle \alpha^* \rangle \Upsilon^+$ implies that $(\nu_n, \nu_{n+1}) \in \rho(\alpha^*)$. Now $(\nu_0, \nu_n) \in \rho(\alpha^*)$ and $(\nu_n, \nu_{n+1}) \in \rho(\alpha^*)$ imply $(\nu_0, \nu_{n+1}) \in \rho(\alpha^*)$. Hence we conclude $\nu_{n+1} \models \psi$ from $\nu_0 \models \phi$ and $\phi \rightarrow [\alpha^*]\psi$. □

By Theorem 1, any formula implying χ_m is also a correct model monitor, such as $\langle \alpha \rangle \Upsilon^+$, which more conservatively limits acceptable executions of the real γ to those that correspond to just one iteration of α^* as opposed to arbitrarily many.

Example 4 (Arithmetical model monitor condition) As illustrated in Fig. 5 and shown concretely below, we can simplify formula (5) into an arithmetical representation $F(x, x^+)$ such that $F(x, x^+) \Rightarrow \langle \alpha^* \rangle \Upsilon^+$, by applying the axioms of dL. The synthesis algorithm to automatically generate the condition $F(x, x^+)$ is presented in Section 3.3.

$$\overbrace{-1 \leq f^+ \leq \tfrac{m-x}{\varepsilon} \wedge x^+ = x + f^+t^+ \wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge f^+t^+ + x \geq 0}^{F(x,x^+)}$$
$$\Rightarrow \langle \underbrace{(f := *;\ ?\left(-1 \leq f \leq \tfrac{m-x}{\varepsilon}\right);\ t := 0;\ \{x' = f,\ t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon\})^*}_{\alpha^*} \rangle \underbrace{(f = f^+ \wedge x = x^+ \wedge t = t^+)}_{\Upsilon^+}$$

The formula $F(x, x^+)$ says that (i) only valid flows should be chosen for the posterior state, i.e., $-1 \leq f^+ \leq \frac{m-x}{\varepsilon}$, (ii) that the posterior water level x^+ must be determined by the prior level x and the flow over time $x^+ = x + f^+t^+$, and (iii) that the evolution domain constraint must be satisfied in both prior and posterior state, i.e., $x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge \underbrace{f^+t^+ + x}_{x^+} \geq 0$.

This formula corresponds to the expected result from Example 3, since x corresponds to $\nu_{i-1}(x)$ and x^+ corresponds to $\nu(x)$, and so forth.

The formula in Example 4 contains checks for water level x, flow f, and time t, because these are the variables changed by the model. If we want to additionally monitor that the model does not change anything except these variables, we can use Corollary 1 to include frame constraints for specific variables into a monitor (e. g., the value of variable ε is not changed by the water tank model, and therefore not supposed to change in reality).

Corollary 1 *Theorem 1 continues to apply when replacing V by any superset $V \supseteq BV(\alpha^*)$.*

Proof Any variable $z \in V \setminus \mathrm{BV}(\alpha^*)$ can be added to Theorem 1 by considering $(z := z; \alpha)^*$ instead of α^*, which has the same behavior but one more bound variable. □

So far, Theorem 1 assumed that everything stays constant, except for the water level x, the flow f, and the time t. This assumption is stronger than absolutely necessary, and, strictly speaking, prevents us from using the monitor in an environment where values that are irrelevant to the model and its safety condition change (e. g., the water temperature). Corollary 2 ensures monitor correctness in environments where irrelevant variables change arbitrarily. Theorem 2 and 3 can be extended with corollaries similar to Corollaries 1 and 2.

Corollary 2 *When replacing V by any superset $V \supseteq BV(\alpha^*) \cup FV([\alpha^*]\psi) \cup FV(\chi_m)$, Theorem 1 continues to hold without the assumption that the ν_k agree on $\Sigma \setminus V$.*

Proof Assume the conditions of Theorem 1 with any sequence of states $\nu_0, \nu_1, \nu_2, \nu_3 \ldots \in \mathbb{R}^n$, with $\nu_0 \models \phi$. Consider a modified sequence of states $\bar{\nu}_0, \bar{\nu}_1, \bar{\nu}_2, \bar{\nu}_3 \ldots$ such that for all k: ν_k agrees with $\bar{\nu}_k$ on V and $\bar{\nu}_k$ agrees with ν_0 on $\Sigma \setminus V$, which, thus, satisfies the assumptions of Theorem 1. Hence, $(\nu_i, \nu_{i+1}) \models \chi_\mathrm{m}$ implies $(\bar{\nu}_i, \bar{\nu}_{i+1}) \models \chi_\mathrm{m}$ by Lemma 2 using $V \supseteq \mathrm{FV}(\chi_\mathrm{m})$. Thus, $(\nu_i, \nu_{i+1}) \models \chi_\mathrm{m}$ for all $i < n$ implies $(\bar{\nu}_i, \bar{\nu}_{i+1}) \models \chi_\mathrm{m}$ for all $i < n$, so Theorem 1 implies $\bar{\nu}_n \models \psi$. Since $V \supseteq \mathrm{FV}(\psi)$, Lemma 2 implies that $\nu_n \models \psi$. □

Theorem 1 ensures that, when the monitor is satisfied, the monitored states are safe, i. e., ψ holds. We can get an even stronger result by Corollary 3, which says that a model monitor also ensures that inductive invariants φ of the model are preserved.

Corollary 3 *Under the conditions of Theorem 1 it is also true that $\nu_n \models \varphi$ for an invariant φ s.t. $\phi \to \varphi$, $\varphi \to [\alpha]\varphi$, and $\varphi \to \psi$.*

Proof From $\models \phi \to [\alpha^*]\psi$ it follows that there exists a φ s.t. $\phi \to \varphi$, $\varphi \to [\alpha]\varphi$, and $\varphi \to \psi$. Hence $\models \phi \to [\alpha^*]\varphi$ and Theorem 1 applies with φ in place of ψ. □

Now that we know the correctness of the logical monitor representation, let us turn to synthesizing its arithmetical form.

3.3 Monitor synthesis algorithm

Our approach to generate monitors from hybrid system models is a correct-by-construction approach. This section explains how to turn monitor specifications into monitor code that can be executed at runtime along with the controller. We take a verified dℒ formula (2) and a synthesis tactic choice (whether to synthesize a model, controller, or prediction monitor) as input and produce a monitor $F(x, x^+)$ in quantifier-free first-order form as output. The algorithm, listed in Algorithm 1, involves the following steps:

1. A dℒ formula (2) about a model α^* of the form $\phi \to [\alpha^*]\psi$ is turned into a specification conjecture (5) of the form $\langle \alpha^* \rangle \Upsilon^+$.

Algorithm 1: ModelPlex monitor synthesis

input : A hybrid program α^*, a set of variables $V \supseteq \mathrm{BV}(\alpha^*)$, a tactic choice τ (model monitor, controller monitor, prediction monitor)
output: A monitor $F(x, x^+)$ that is first-order and implies $\langle\alpha^*\rangle\Upsilon^+$
begin
  $\Upsilon^+ \longleftarrow \bigwedge_{x\in V} x = x^+$ with fresh variables x_i^+ // Specification conjecture
  $G \longleftarrow \{\langle\alpha^*\rangle\Upsilon^+\}$ // Set of proof goals
  $S \longleftarrow \emptyset$ // Specification goals
1 **while** $G \neq \emptyset$ **do** // Analyze specification conjecture
    choose any goal $g \in G$
    $G \longleftarrow G \setminus \{g\}$
    if *g is first-order* **then**
      if $\not\models g$ **then** $S \longleftarrow S \cup \{g\}$ // Monitor at runtime if unprovable
    else
      $\tilde{g} \longleftarrow$ apply $\mathsf{d}\mathscr{L}$ proof rule according to tactic τ to g // Simplification step
      $G \longleftarrow G \cup \{\tilde{g}\}$
  $F(x, x^+) \longleftarrow \bigwedge_{s\in S} s$ // Collect all open goals

2. Theorem proving according to the tactic choice is applied on the specification conjecture (5) until no further $\mathsf{d}\mathscr{L}$ proof rules are applicable and only first-order real arithmetic formulas remain open.
3. The monitor specification $F(x, x^+)$ is the conjunction of the unprovable first-order real arithmetic formulas from open sub-goals. The intuition behind this is that goals, which remain open in the offline proof, are proved online through monitoring. Although this do not yield a proof for all imaginable runs, that way we obtain a *proof for the current run* of the real CPS.

The correctness of the monitoring conditions obtained through Algorithm 1 is guaranteed by the soundness of the $\mathsf{d}\mathscr{L}$ calculus. In the remainder of the section, we will exemplify Algorithm 1 by turning the model of the water tank example into a model monitor.

Generate the specification conjecture We map $\mathsf{d}\mathscr{L}$ formula (2) syntactically to a specification conjecture of the form (5), i. e., $\langle\alpha^*\rangle\Upsilon^+$. By design, this conjecture will not be provable. But the unprovable branches of a proof attempt will reveal information that, had it been in the premises, would make (5) provable. Through Υ^+, those unprovable conditions collect the relations of the posterior state of model α^* characterized by x^+ to the prior state x, i. e., the conditions are a representation of (4) in quantifier-free first-order real arithmetic.

Example 5 (Specification conjecture) The specification conjecture for the water tank model monitor is:

$$\langle\left(f := *;\, ?\left(-1 \leq f \leq \tfrac{m-x}{\varepsilon}\right);\; t := 0;\;\; \left\{x' = f,\; t' = 1 \;\&\; x \geq 0 \wedge t \leq \varepsilon\right\}\right)^*\rangle \overbrace{(x = x^+ \wedge f = f^+ \wedge t = t^+)}^{\Upsilon^+}$$

It is constructed by Algorithm 1 in steps "specification conjecture" and "set of proof goals" from the model by flipping the modality and formulating the specification requirement as a property, since we are interested in a relation between two consecutive states ν and ω (recalled by x^+, f^+ and t^+).

Use theorem proving to analyze the specification conjecture We use the axioms and proof rules of $\mathsf{d}\mathscr{L}$ [29,33,35] to analyze the specification conjecture $\langle\alpha^*\rangle\Upsilon^+$. These proof rules

syntactically decompose a hybrid model into easier-to-handle parts, which leads to sequents with first-order real arithmetic formulas towards the leaves of a proof. Using real arithmetic quantifier elimination we close sequents with logical tautologies, which do not need to be checked at runtime since they always evaluate to *true* for any input. The conjunction of the remaining open sequents is the monitor specification; it implies formula (4).

In the remainder of this article, we follow a synthesis style based on the axiomatization of $\mathsf{d}\mathscr{L}$. Axiomatization-style synthesis differs from the sequent-style synthesis of the short version [24] in the mechanics of the simplification step of Algorithm 1. The axiomatization of $\mathsf{d}\mathscr{L}$ allows working in place with fast contextual congruences. This leads to simpler monitors and simpler proofs since the synthesis proof does not branch and thus keeps working on the same goal ($\tilde{g} = g$, so $|G| = 1$), as opposed to the sequent-style synthesis, which may create new goals ($|G| \geq 1$). For comparison, the corresponding sequent-style synthesis techniques of the short version [24] of this article is elaborated in Appendix 3. The complete proof calculus is reported in the literature [29,33,35]. We explain the requisite proof rules on-the-fly while discussing their use in the running example.

Example 6 (Analyzing loops, assignments, and tests) The analysis of the water tank conjecture from Example 5 uses $\langle * \rangle$ elim to eliminate the loop, $\langle ; \rangle$ to handle the sequential composition, followed by $\langle := * \rangle$ to analyze the nondeterministic assignment $\langle f := * \rangle$. The hybrid program *plant* is an abbreviation for $t := 0;\ \{x' = f, t' = 1 \,\&\, x \geq 0 \wedge t \leq \varepsilon\}$, whereas Υ^+ is an abbreviation for $x = x^+ \wedge f = f^+ \wedge t = t^+$. The nondeterministic assignment axiom $\langle := * \rangle$ introduces an existential quantifier. Note that rewriting can still continue in-place, as demonstrated by rewriting the sequential composition and test inside the quantifier.

$$(\langle ; \rangle)\quad \langle \alpha; \beta \rangle \phi \leftrightarrow \langle \alpha \rangle \langle \beta \rangle \phi \qquad (\langle := * \rangle)\quad \langle x := * \rangle \phi(x) \leftrightarrow \exists x\, \phi(x)$$

$$(\langle ? \rangle)\quad \langle ?H \rangle \psi \leftrightarrow (H \wedge \psi) \qquad (\langle * \rangle \text{ elim})\quad \langle \alpha \rangle \phi \rightarrow \langle \alpha^* \rangle \phi$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \exists f \left(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \langle plant \rangle \Upsilon^+\right)}{\vdash \exists f \langle ? -1 \leq f \leq \frac{m-x}{\varepsilon} \rangle \langle plant \rangle \Upsilon^+}\langle ? \rangle}{\vdash \exists f \langle ? -1 \leq f \leq \frac{m-x}{\varepsilon}; plant \rangle \Upsilon^+}\langle ; \rangle}{\vdash \langle f := *; ? -1 \leq f \leq \frac{m-x}{\varepsilon}; plant \rangle \Upsilon^+}\langle := * \rangle}{\vdash \langle f := *; ? -1 \leq f \leq \frac{m-x}{\varepsilon}; plant \rangle \Upsilon^+}\langle ; \rangle}{\vdash \langle \left(f := *; ? -1 \leq f \leq \frac{m-x}{\varepsilon}; plant\right)^* \rangle \Upsilon^+}\langle * \rangle \text{ elim}$$

Let us look more closely into the first step of Example 6, i. e., $\langle * \rangle$ elim. Usually, proving properties of the form $\langle \alpha^* \rangle \phi$ about loops requires an *inductive variant* in order to prove arbitrarily many repetitions of the loop body. With monitoring in mind, though, we can *unwind* the loop and execute the resulting conditions repeatedly instead, as elaborated in Lemma 5.

Lemma 5 *(Loop elimination) Let α be a hybrid program and α^* be the program that repeats α arbitrarily many times. Then $\langle \alpha \rangle \phi \rightarrow \langle \alpha^* \rangle \phi$ is valid.*

Proof We prove in $\mathsf{d}\mathscr{L}$ using loop unwinding $\langle * \rangle$, monotonicity [] mon and propositional reasoning as follows.

$$((*)) \quad \langle\alpha^*\rangle\phi \leftrightarrow \phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi \qquad ([]\ \text{mon}) \quad \frac{\phi \vdash \psi}{[\alpha]\phi \vdash [\alpha]\psi} \qquad (\to\text{l}) \quad \frac{\Gamma \vdash \phi, \Delta \quad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \to \psi \vdash \Delta}$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \langle\alpha\rangle\phi}\,{\to}\text{l,ax}}{\langle\alpha\rangle\phi \to \langle\alpha\rangle(\phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi) \vdash \phi \vee \langle\alpha\rangle(\phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi)}\,\text{cut} \qquad \dfrac{\dfrac{*}{\vdash \phi \to \phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi}}{\vdash \langle\alpha\rangle\phi \to \langle\alpha\rangle(\phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi)}\,[]\ \text{mon}}{\vdash \phi \vee \langle\alpha\rangle(\phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi)}\,\langle*\rangle$$

$$\dfrac{\vdash \phi \vee \langle\alpha\rangle(\phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi)}{\vdash \phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi}\,\langle*\rangle$$

$$\dfrac{\vdash \phi \vee \langle\alpha\rangle\langle\alpha^*\rangle\phi}{\vdash \langle\alpha^*\rangle\phi}$$

□

Lemma 5 allows us to check compliance with the model α^* by checking compliance on each execution of α (i.e., online monitoring [18]), which is easier than for α^* because the loop was eliminated.

We will continue Example 6 in subsequent examples. The complete sequence of proof rules applied to the specification conjecture of the water tank is described in Appendix 2. Most steps are simple when analyzing specification conjectures: sequential composition ($\langle;\rangle$), nondeterministic choice ($\langle\cup\rangle$), deterministic assignment ($\langle:=\rangle$) replace current facts with simpler ones (or branch the proof as propositional rules do). Challenges arise from handling nondeterministic assignment and differential equations in hybrid programs.

Let us first consider nondeterministic assignment $x := *$. The proof rule for nondeterministic assignment ($\langle:=*\rangle$) results in a new existentially quantified variable. Using axiomatic-style synthesis, we can postpone instantiating the quantifier until enough information about what exact instance to use is discovered, see Example 7. The sequent-style synthesis, in contrast, must instantiate the quantifier right away, in order to continue synthesis on the existentially quantified formula. Appendix 3 discusses ways on how to instantiate such quantifiers ahead of time.

Next, we handle differential equations. Even when we can solve the differential equation, existentially and universally quantified variables remain. Let us inspect the corresponding proof rule from the $\mathsf{d}\mathscr{L}$ calculus [33] in its axiomatic form.

$$(\langle'\rangle) \quad \big(y(0) = x \wedge [T' = 1]y(T)' = \theta(y(T))\big) \to \Big(\langle x' = \theta\ \&\ H\rangle\phi \leftrightarrow \exists T{\geq}0\ \big((\forall 0{\leq}\tilde{t}{\leq}T\ \langle x := y(\tilde{t})\rangle H) \wedge \langle x := y(T)\rangle\phi\big)\Big)^{1} \qquad (\text{QE}) \quad \frac{\text{QE}(\phi)^{2}}{\phi}$$

[1] T and $\tilde{t}$ are fresh logical variables

[2] iff $\phi \equiv \text{QE}(\phi)$, ϕ is a first-order real arithmetic formula, $\text{QE}(\phi)$ is an equivalent quantifier-free formula computable by [7]

When solving differential equations, we first have to prove the correctness of the solution, as indicated by the left-hand side of the implication in axiom $\langle'\rangle$. Then, we have to prove that there exists a duration T, such that the differential equation stays within the evolution domain H throughout all intermediate times $\tilde{t}$ and the result satisfies ϕ at the end. At this point we have four options:

- we can postpone handling the quantifier until additional facts about a concrete instance are discovered, which is the preferred tactic in axiomatic-style synthesis;
- we can instantiate the existential quantifier, if we know that the duration will be t^+;

– we can introduce a new logical variable, which is the generic case in sequent-style synthesis that always yields correct results, but may discover monitor specifications that are harder to evaluate;
– we can use quantifier elimination (QE) to obtain an equivalent quantifier-free result (a possible optimization could inspect the size of the resulting formula).

Example 7 (Analyzing differential equations) Continuing Example 6, in the analysis of the water tank example, we solve the differential equation, see $\langle' \rangle$. The condition $y(0) = x \wedge [T' = 1]y(T)' = \theta(y(T))$, with the solution $y(T) = fT + x$ of this example, is closed on a side branch. Next, we have an existential quantifier with an equality $t = 0$, so we can instantiate t with 0 by $\exists\sigma$. In the next step, we instantiate the existential quantifier $\exists T$ with t^+, as now revealed in the last conjunct $t^+ = T$; we do the same for $\exists f$ by $f = f^+$. Finally, we use quantifier elimination (QE) to reveal an equivalent quantifier-free formula.

$$(\exists\sigma)\quad \frac{\phi(\theta)}{\exists x = \theta\ \phi(x)}\ {}^{1}$$

[1] Logical variable x does not appear in term θ

$$\begin{array}{ll}
 & \vdash -1 \le f^+ \le \frac{m-x}{\varepsilon} \wedge x^+ = x + f^+t^+ \wedge x \ge 0 \wedge \varepsilon \ge t^+ \ge 0 \wedge f^+t^+ + x \ge 0 \\
\text{QE} & \vdash -1 \le f^+ \le \frac{m-x}{\varepsilon} \wedge \forall 0 \le \tilde{t} \le t^+ \ (x + f^+\tilde{t} \ge 0 \wedge \tilde{t} \le \varepsilon) \wedge f^+ = f^+ \wedge x^+ = x + f^+t^+ \wedge t^+ = t^+ \\
\exists\sigma & \vdash \exists f \ \left(-1 \le f \le \frac{m-x}{\varepsilon} \wedge \forall 0 \le \tilde{t} \le t^+ \ (x + f\tilde{t} \ge 0 \wedge \tilde{t} \le \varepsilon) \wedge f = f^+ \wedge x^+ = x + ft^+ \wedge t^+ = t^+\right) \\
\exists\sigma & \vdash \exists f \ \left(-1 \le f \le \frac{m-x}{\varepsilon} \wedge \exists T \ge 0 \left(\forall 0 \le \tilde{t} \le T \ (x + f\tilde{t} \ge 0 \wedge \tilde{t} \le \varepsilon) \wedge f = f^+ \wedge x^+ = x + fT \wedge t^+ = T\right)\right) \\
\exists\sigma & \vdash \exists f \ \left(-1 \le f \le \frac{m-x}{\varepsilon} \wedge \exists t = 0 \exists T \ge 0 (\forall 0 \le \tilde{t} \le T \ (x + f\tilde{t} \ge 0 \wedge \tilde{t} + t \le \varepsilon)\right. \\
 & \qquad \left. \wedge f = f^+ \wedge x^+ = x + fT \wedge t^+ = T + t)\right) \\
\langle' \rangle & \vdash \exists f \ \left(-1 \le f \le \frac{m-x}{\varepsilon} \wedge \exists t = 0 \langle \{x' = f, t' = 1 \ \&\ x \ge 0 \wedge t \le \varepsilon\} \rangle \Upsilon^+\right) \\
\langle;\rangle,\langle:=\rangle & \vdash \exists f \ \left(-1 \le f \le \frac{m-x}{\varepsilon} \wedge \langle t := 0;\ \{x' = f, t' = 1 \ \&\ x \ge 0 \wedge t \le \varepsilon\} \rangle \Upsilon^+\right)
\end{array}$$

The analysis of the specification conjecture finishes with collecting the open sequents from the proof to create the monitor specification $F(x, x^+) \overset{\text{def}}{\equiv} \bigwedge(\textit{open sequent})$. The axiomatic-style synthesis operates fully in-place, so there is only one open sequent to collect. In contrast, the sequent-style synthesis usually splits into multiple branches. Moreover, the collected open sequents may include new logical variables and new (Skolem) function symbols that were introduced for nondeterministic assignments and differential equations when handling existential or universal quantifiers. These can be handled in a final step by re-introducing and instantiating quantifiers, see Appendix 3.

Let us now recall our desired result from Example 3 and compare it to the formula synthesized in Examples 6 and 7. Also recall that ν_{i-1} denotes the prior state and ν_i denotes the posterior state of running the model, so we have the following correlations of symbols: $\nu_{i-1}(f)$ corresponds to f, $\nu_{i-1}(t)$ to t, $\nu_{i-1}(x)$ to x, whereas $\nu_i(f)$ corresponds to f^+, $\nu_i(t)$ to t^+, and $\nu_i(x)$ to x^+.

$$\underbrace{-1 \le \nu_i(f) \le \frac{m - \nu_{i-1}(x)}{\varepsilon}}_{-1 \le f^+ \le \frac{m-x}{\varepsilon}} \wedge \underbrace{\nu_i(x) = \nu_{i-1}(x) + \nu_i(f)\nu_i(t)}_{x^+ = x + f^+t^+} \wedge \underbrace{\nu_{i-1}(x) \ge 0}_{x \ge 0} \wedge \underbrace{\nu_i(x) \ge 0}_{f^+t^+ + x \ge 0} \wedge \underbrace{0 \le \nu_i(t) \le \varepsilon}_{\varepsilon \ge t^+ \ge 0}$$

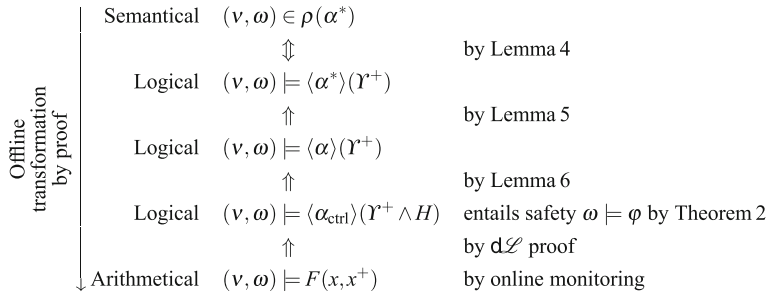


Fig. 7 Semantical representation, logical characterization, and arithmetical form of a controller monitor. Monitor synthesis translates between these representations offline

The conjuncts from the synthesized formula cover all the desired conditions nicely, considering that x^+ is expanded to its lengthier equal form $x^+ = x + f^+t^+$.

Remark 1 (Monitor evaluation at runtime) The complexity of evaluating an arithmetic formula over the reals for concrete numbers (such as a monitor for the concrete numbers corresponding to the current state) is linear in the formula size, as opposed to deciding the validity of such formulas, which is doubly exponential [10]. Evaluating the same formula on floating point numbers is inexpensive, but may yield incorrect results due to rounding errors; on exact rationals the bit-complexity can be non-negligible. We use interval arithmetic to obtain correct results while retaining the efficiency of floating-point computations. Interval arithmetic over-approximates a real value using an interval of two floating-point values that contains the real, which means the monitors become more conservative (e. g., to evaluate $x \leq m$ in interval arithmetic, consider $x \in [x_l, x_u]$ and $m \in [m_l, m_u]$, so $[x_l, x_u] \leq [m_l, m_u]$ if $x_u \leq m_l$, which in turn implies $x \leq m$). This leads to an interval-arithmetic formula $\hat{F}(x, x^+)$ that implies $F(x, x^+)$ and, thus, also implies the required monitor condition Formula (5).

3.4 Controller monitor synthesis

For a hybrid system α^* of the canonical form $(\alpha_{\text{ctrl}}; \alpha_{\text{plant}})^*$, a *controller monitor* χ_c, cf. Fig. 8, checks that two consecutive states ν and $\tilde{\nu}$ are reachable with one controller execution α_{ctrl}, i. e., $(\nu, \tilde{\nu}) \in \rho(\alpha_{\text{ctrl}})$ with $V = \text{BV}(\alpha_{\text{ctrl}})$. This controller monitor is to be executed before a control choice by the controller is sent to the actuators. The program α_{ctrl} is derived from α by skipping differential equations according to Lemma 6 below. Recall that a differential equation $\{x' = \theta \;\&\; H\}$ can be followed for a nondeterministic amount of time, including 0, which lets us skip it as long as its evolution domain constraint H is satisfied in the beginning, as captured by $\langle\alpha_{\text{ctrl}}\rangle(\Upsilon^+ \wedge H)$. That way, a controller monitor ensures that the states reachable by a controller enable subsequent runs of the plant, see Theorem 2. We systematically derive a controller monitor from the specification formula $\langle\alpha^*\rangle\Upsilon^+$, see Fig. 7.

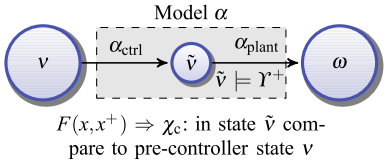


Fig. 8 A controller monitor checks that two states ν and $\tilde{\nu}$ are contained in the transition relation of the controller portion of the model $(\nu, \tilde{\nu}) \in \rho(\alpha_{\text{ctrl}})$; the posterior state $\tilde{\nu}$ is captured in x^+ through $\Upsilon+$

A controller monitor can be used to initiate controller switching similar to Simplex [39], yet in provably correct-by-construction ways.

Lemma 6 *(Differential skip) Let $x' = \theta$ denote a set of differential equations with evolution domain H. Then $H \wedge \phi \rightarrow \langle x' = \theta \,\&\, H\rangle\phi$ is valid.*

Proof We prove in $\mathsf{d}\mathscr{L}$ using [′] skip derived from DW [35].

$$(\neg\neg)\ \neg\neg\phi \leftrightarrow \phi \qquad (\neg\wedge)\ \neg(\phi \wedge \psi) \leftrightarrow \neg\phi \vee \neg\psi \qquad (\rightarrow)\ (\phi \rightarrow \psi) \leftrightarrow \neg\phi \vee \psi$$

$$([']\ \text{skip})\ [x' = \theta \,\&\, H]\phi \rightarrow (H \rightarrow \phi) \qquad (\langle\cdot\rangle)\ \langle a\rangle\phi \leftrightarrow \neg[a]\neg\phi$$

$$\dfrac{\dfrac{\dfrac{\dfrac{*}{\vdash \langle x' = \theta \,\&\, H\rangle\phi \rightarrow \langle x' = \theta \,\&\, H\rangle\phi}}{\vdash \neg[x' = \theta \,\&\, H](\neg\phi) \rightarrow \langle x' = \theta \,\&\, H\rangle\phi}\ \langle\cdot\rangle}{\vdash \neg(H \rightarrow \neg\phi) \rightarrow \langle x' = \theta \,\&\, H\rangle\phi}\ [']\ \text{skip}}{\vdash H \wedge \phi \rightarrow \langle x' = \theta \,\&\, H\rangle\phi}\ \neg\neg,\neg\wedge,\rightarrow$$

□

Theorem 2 *(Controller monitor correctness) Let α be of the canonical form $\alpha_{ctrl}; \alpha_{plant}$ with the continuous model $\alpha_{plant} \equiv x' = \theta \,\&\, H$ and let $V = BV(\alpha_{ctrl})$. Assume $\models \phi \rightarrow [\alpha^*]\psi$ has been proven with invariant φ as in (3), i.e., $\phi \rightarrow \varphi$, $\varphi \rightarrow [\alpha]\varphi$, and $\varphi \rightarrow \psi$. Let $\nu \models \varphi$, as is checked by χ_m (Corollary 3). Furthermore, let $\tilde{\nu}$ be the state after running the actual CPS controller implementation and let $\tilde{\nu}$ agree with ν on $\Sigma\backslash V$, i.e., $\nu|_{\Sigma\backslash V} = \tilde{\nu}|_{\Sigma\backslash V}$. If $(\nu, \tilde{\nu}) \models \chi_c$ with*

$$\chi_c \equiv \langle\alpha_{ctrl}\rangle(\Upsilon^+ \wedge H)$$

then $(\nu, \tilde{\nu}) \in \rho(\alpha_{ctrl})$, $\tilde{\nu} \models \varphi$, and there exists a state ω such that $(\tilde{\nu}, \omega) \in \rho(\alpha_{plant})$.

Proof By Lemma 4, $(\nu, \tilde{\nu}) \models \langle\alpha_{\text{ctrl}}\rangle\Upsilon^+$ implies $(\nu, \tilde{\nu}) \in \rho(\alpha_{\text{ctrl}})$. The assumption $(\nu, \tilde{\nu}) \models \langle\alpha_{\text{ctrl}}\rangle(\Upsilon^+ \wedge H)$ furthermore implies $\tilde{\nu} \models H$, $(\nu, \tilde{\nu}) \models \langle\alpha_{\text{ctrl}}; \alpha_{\text{plant}}\rangle\Upsilon^+$ by Lemma 6, hence $(\nu, \tilde{\nu}) \models \langle\alpha\rangle\Upsilon^+$ and $(\nu, \tilde{\nu}) \in \rho(\alpha)$ by Lemma 4. Since $\nu \models \varphi$ by assumption, we get $\tilde{\nu} \models \varphi$ from $\varphi \rightarrow [\alpha]\varphi$. Now $\tilde{\nu} \models \varphi \wedge H$, so there exists ω s.t. $(\tilde{\nu}, \omega) \in \rho(\alpha_{\text{plant}})$. □

The corollaries to Theorem 1 carry over to Theorem 2 accordingly.

3.5 Monitoring in the presence of expected uncertainty and disturbance

Up to now we considered exact ideal-world models. But real-world clocks drift, sensors measure with some uncertainty, and actuators are subject to disturbance. This makes the exact models safe but too conservative, which means that monitors for exact models are likely to fall back to a fail-safe controller rather often. In this section we discuss how we find ModelPlex specifications in the sequent-style synthesis techniques so that the safety property (2) and the monitor specification become more robust to expected uncertainty and disturbance. That way, only unexpected deviations beyond those captured in the normal operational uncertainty and disturbance of model α^* cause the monitor to initiate fail-safe actions.

In $\mathsf{d}\mathscr{L}$, we can, for example, use nondeterministic assignment from an interval to model sensor uncertainty and piece-wise constant actuator disturbance (e. g., as in [26]),

or differential inequalities for actuator disturbance (e. g., as in [38]). Such models include nondeterminism about sensed values in the controller model and often need more complex physics models than differential equations with polynomial solutions.

Example 8 (Modeling uncertainty and disturbance) We incorporate clock drift, sensor uncertainty and actuator disturbance into the water tank model to express expected deviation. The measured level x_s is within a known sensor uncertainty u of the real level x (i.e. $x_s \in [x-u, x+u]$). We use differential inequalities to model clock drift and actuator disturbance. The clock, which wakes the controller, is slower than the real time by at most a time drift of c; it can be arbitrarily fast. The water flow disturbance is at most d, but the water tank is allowed to drain arbitrarily fast (may even leak when the outgoing valve is closed). To illustrate different modeling possibilities, we use additive clock drift and multiplicative actuator disturbance.

$$0 \le x \le m \land \varepsilon > 0 \land\ c < 1 \land 0 \le u \land 0 < d$$
$$\to \Big[\big(x_s := *; ?\,(x-u \le x_s \le x+u)\,;\ f := *; ?\,\big(-1 \le f \le \tfrac{m-x_s-u}{d\varepsilon}(1-c)\big);$$
$$t := 0;\ \{x' \le fd,\ 1-c \le t' \,\&\, x \ge 0 \land t \le \varepsilon\}\big)^* \Big](0 \le x \le m)$$

We analyze Example 8 in the same way as the previous examples, with the crucial exception of the differential inequalities. We cannot use the proof rule $\langle' \rangle$ to analyze this model, because differential inequalities do not have polynomial solutions. Instead, we use DM (cf. Lemma 7) and the DE proof rule of $\mathsf{d}\mathscr{L}$ [31] to turn differential inequalities into a differential-algebraic constraint form that lets us proceed with the proof. Rule DE turns a differential inequality $x' \le \theta$ into a quantified differential equation $\exists \tilde{d}(x' = \tilde{d} \,\&\, \tilde{d} \le \theta)$ with an equivalent differential-algebraic constraint. Rule DM turns a fluctuating disturbance $\langle \exists \tilde{d}\ x' = \tilde{d} \,\&\, H \rangle \phi$ into a mean disturbance $\exists \bar{d} \langle x' = \bar{d} \,\&\, H \rangle \phi$, see Lemma 7.

Lemma 7 *(Mean disturbance) Reachability with mean disturbance $\bar{d}$ throughout approximates fluctuating disturbance $\tilde{d}$: (DM)* $\quad \exists \bar{d} \langle x' = \theta(\bar{d}) \,\&\, H \rangle \phi \to \langle \exists \tilde{d}\ x' = \theta(\tilde{d}) \,\&\, H \rangle \phi$.

Proof We prove in $\mathsf{d}\mathscr{L}$ using differential refinement DR [31].

$$\text{(DR)}\ \ \frac{\mathscr{D} \to \mathscr{E} \quad \langle \mathscr{D} \rangle \phi}{\langle \mathscr{E} \rangle \phi}\ {}^{1} \qquad \text{(}\exists\text{l)}\ \ \frac{\Gamma, \phi(s(X_1, \ldots, X_n)) \vdash \Delta}{\Gamma, \exists x\, \phi(x) \vdash \Delta}\ {}^{2} \qquad \text{(}\exists\text{inst)}\ \ p(t()) \to \exists x\, p(x)$$

[1] differential refinement: differential-algebraic constraints $\mathscr{D}, \mathscr{E}$ have the same changed variables

[2] s is a new (Skolem) function symbol and $X_1, \ldots, X_n$ are all free variables of $\exists x\, \phi(x)$

$$\dfrac{\dfrac{\overset{*}{\overbrace{\vdash \forall x, x'(x' = \theta(\bar{d}())\ \&\ H \to \exists \tilde{d}\,(x' = \theta(\tilde{d})\ \&\ H))}}\ {}^{\exists\text{inst,ax}} \qquad \overset{*}{\overbrace{\langle x' = \theta(\bar{d}())\ \&\ H \rangle \phi \vdash \langle x' = \theta(\bar{d}())\ \&\ H \rangle \phi}}\ {}^{\text{ax}}}{\langle x' = \theta(\bar{d}())\ \&\ H \rangle \phi \vdash \langle \exists \tilde{d}\ x' = \theta(\tilde{d})\ \&\ H \rangle \phi}\ {}^{\text{DR}}}{\exists \bar{d} \langle x' = \theta(\bar{d})\ \&\ H \rangle \phi \vdash \langle \exists \tilde{d}\ x' = \theta(\tilde{d})\ \&\ H \rangle \phi}\ {}^{\exists\text{l}}$$

Example 9 (Analyzing differential inequalities) Loops, assignments and tests are analyzed as in the previous examples. We continue with differential inequalities as follows. First, we eliminate the differential inequalities by rephrasing them as differential-algebraic constraints in step (DE). Then, we refine by extracting the existential quantifiers for flow disturbance $\tilde{d}$ and time drift $\tilde{t}$, so that they become mean disturbance and mean time drift in step (DM). Note, that the existential quantifier moved from inside the modality $\langle \exists \tilde{d}(x' = \tilde{d}\rangle, \ldots)$ to the outside $\exists \bar{d} \langle x' = \bar{d},\ \ldots \rangle$, which captures that the states reachable with fluctuating disturbance could

also have been reached by following a mean disturbance throughout. The resulting differential equation has polynomial solutions and, thus, we can use $\langle' \rangle$ and proceed with the proof as before.

$$\text{(DM)}\ \exists \bar{d} \langle x' = \theta(\bar{d}) \,\&\, H \rangle \phi \rightarrow \langle \exists \tilde{d}\, x' = \theta(\tilde{d}) \,\&\, H \rangle \phi$$

$$\text{(DE)}\ \frac{\begin{array}{c}\forall X (\exists \tilde{d} (X = \tilde{d} \wedge \tilde{d} \leq \theta \wedge H) \rightarrow X \leq \theta \wedge H) \\ \langle \exists \tilde{d} (x' = \tilde{d} \,\&\, \tilde{d} \leq \theta \wedge H) \rangle \phi\end{array}}{\langle x' \leq \theta \,\&\, H \rangle \phi}\ {}^{1}$$

[1] differential inequality elimination: special case of DR, which rephrases the differential inequalities $\leq$ as differential-algebraic constraints (accordingly for other or mixed inequalities systems).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \forall x,t (\exists \tilde{d},\tilde{t} (x = \tilde{d} \wedge t = \tilde{t} \wedge \tilde{d} \leq fd \wedge 1 - c \leq \tilde{t} \wedge \tilde{d} \geq 0 \wedge \tilde{t} \leq \varepsilon) \rightarrow x \leq fd \wedge 1 - c \leq t \wedge x \geq 0 \wedge t \leq \varepsilon)}\,{}^{*}\ \text{QE}}{\vdash \ldots}}{\vdash \exists x_s, f, t\, (\ldots \wedge \exists \bar{d},\bar{t} \langle x' = \bar{d}, t' = \bar{t} \,\&\, \bar{d} \leq fd \wedge 1 - c \leq \bar{t} \wedge x \geq 0 \wedge t \leq \varepsilon \rangle \Upsilon^{+})}\ \langle' \rangle, \text{Opt. 1}, \text{QE}}{\vdash \exists x_s, f, t\, (\ldots \wedge \langle \exists \tilde{d},\tilde{t} (x' = \tilde{d}, t' = \tilde{t} \,\&\, \tilde{d} \leq fd \wedge 1 - c \leq \tilde{t} \wedge x \geq 0 \wedge t \leq \varepsilon \rangle \Upsilon^{+})}\ \text{DM}}{\vdash \exists x_s, f, t\, (\ldots \wedge \langle x' \leq fd, 1 - c \leq t' \,\&\, x \geq 0 \wedge t \leq \varepsilon \rangle \Upsilon^{+})}\ \text{DE}}{\vdash \ldots \text{(analyze as in previous examples)}}$$

$$\dfrac{}{\vdash \langle (x_s := *;\ \ldots;\ \{x' \leq fd,\ 1 - c \leq t' \,\&\, x \geq 0 \wedge t \leq \varepsilon\})^{*} \rangle \Upsilon^{+}}\ \langle * \rangle$$

As expected, we get a more permissive monitor specification. Such a monitor specification says that there exists a mean disturbance $\bar{d}$ and a mean clock drift $\bar{c}$ within the allowed disturbance bounds, such that the measured flow f^+, the clock t^+, and the measured level x^+ can be explained with the model. These existential quantifiers will be turned into equivalent quantifier-free form in subsequent steps by QE.

So far, we discussed proof rule $\langle' \rangle$ to solve differential equations when synthesizing model monitors. Recent advances [41] on proving $\langle \cdot \rangle \phi$ properties (where ϕ is phrased using equalities) point to an interesting direction for synthesizing model monitors without solving differential equations. In the next section, we will use $\mathsf{d}\mathscr{L}$ techniques based on differential invariants, differential cuts [30], and differential auxiliaries [32] to handle differential equations and inequalities without requiring any closed-form solutions when synthesizing prediction monitors.

3.6 Monitoring compliance guarantees for unobservable intermediate states

With controller monitors, non-compliance of a controller implementation w.r.t. the modeled controller can be detected right away. With model monitors, non-compliance of the actual system dynamics w.r.t. the modeled dynamics can be detected when they first occur. We switch to a fail-safe action, which is verified using standard techniques, in both non-compliance cases. The crucial question is: can such a method always guarantee safety? The answer is linked to the image computation problem in model checking (i. e., approximation of states reachable from a current state), which is known to be not semi-decidable by numerical evaluation at points; approximation with uniform error is only possible if a bound is known for the continuous derivatives [36]. This implies that we need additional assumptions about the deviation between the actual and the modeled continuous dynamics to guarantee compliance for unobservable intermediate states. Unbounded deviation from the model between sample

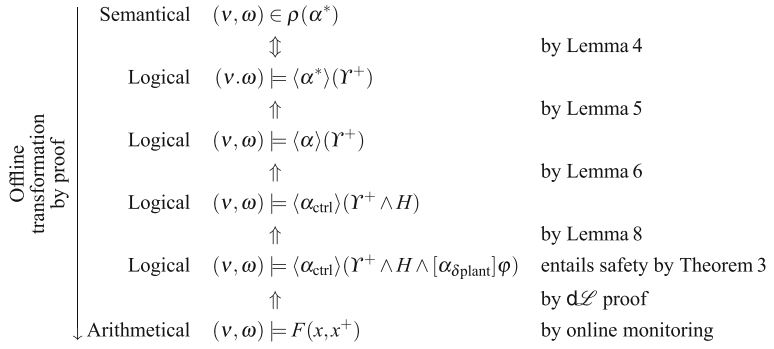


Fig. 9 Semantical representation, logical characterization, and arithmetical form of a prediction monitor. Monitor synthesis translates between these representations offline

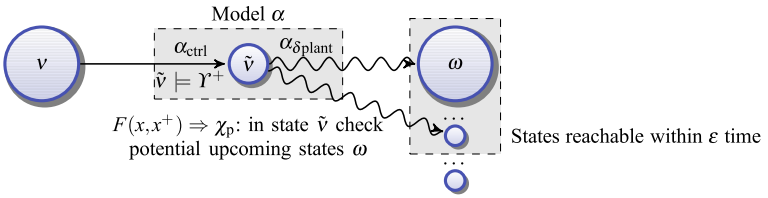


Fig. 10 A prediction monitor checks that none of the potential states ω reachable from state $\tilde{\nu}$ by following the plant with some disturbance δ for up to time ε is unsafe; the posterior state $\tilde{\nu}$ is captured in x^+ through Υ^+

points just is unsafe, no matter how hard a controller tries. Hence, worst-case bounds capture how well reality is reflected in the model.

We derive a prediction monitor, cf. Figs. 9 and 10, to check whether a current control decision will be able to keep the system safe for time ε even if the actual continuous dynamics deviate from the model. A prediction monitor checks the current state, because all previous states are ensured by a model monitor and subsequent states are then safe by (2).

In order to derive a prediction monitor, we use Lemma 8 to introduce a plant with disturbance as additional predicate into our logical representation.

Lemma 8 *(Introduce predicate) Formula* $\langle\alpha\rangle(\phi \wedge \psi) \rightarrow \langle\alpha\rangle\phi$ *is valid.*

Proof Follows from $\phi \wedge \psi \rightarrow \phi$ using the diamond variant of [] mon. □

Definition 4 (ε-bounded plant with disturbance δ) Let α_{plant} be a model of the form $x' = \theta \,\&\, H$. An *ε-bounded plant with disturbance δ*, written $\alpha_{\delta\text{plant}}$, is a plant model of the form $x_0 := 0;\ \big(f(\theta,\delta) \le x' \le g(\theta,\delta) \,\&\, H \wedge x_0 \le \varepsilon\big)$ for some f, g with fresh variable $\varepsilon > 0$ and with a clock $x_0' = 1$. We say that disturbance δ is *constant* if $x \notin \text{FV}(\delta)$; it is *additive* if $f(\theta,\delta) = \theta - \delta$ and $g(\theta,\delta) = \theta + \delta$.

Theorem 3 *(Prediction monitor correctness) Let α be of the canonical form $\alpha_{ctrl}; \alpha_{plant}$ with the continuous model $\alpha_{plant} \equiv x' = \theta \,\&\, H$ and let $V = BV(\alpha) \cup FV([\alpha_{\delta plant}]\varphi)$. Let α^* be provably safe, i.e., $\models \phi \rightarrow [\alpha^*]\psi$ has been proved using invariant φ as in (3). Let $\nu \models \varphi$, as checked by χ_m from Corollary 3. If $(\nu,\tilde{\nu}) \models \chi_p$ with*

$$\chi_p \equiv \langle\alpha_{ctrl}\rangle(\Upsilon^+ \wedge H \wedge [\alpha_{\delta plant}]\varphi)$$

then we have $\omega \models \varphi$ for all ω s.t. $(\nu,\omega) \in \rho(\alpha_{ctrl}; \alpha_{\delta plant})$.

Proof Assume $(\nu, \tilde{\nu}) \models \chi_p$, i.e., $\nu_{x^+}^{\tilde{\nu}(x)} \models \chi_p$. By Theorem 2, $(\nu, \tilde{\nu}) \models \langle \alpha_{ctrl} \rangle \Upsilon^+$ implies $\tilde{\nu} \models \varphi$, since $\nu \models \varphi$. Furthermore, then there exists μ such that $\mu \models \Upsilon^+ \wedge H \wedge [\alpha_{\delta plant}]\varphi$ with $(\nu_{x^+}^{\tilde{\nu}(x)}, \mu) \in \rho(\alpha_{ctrl})$ and the two states ν and μ agree on all variables except the ones modified by α_{ctrl}, i.e., $\nu_{x^+}^{\tilde{\nu}(x)}|_{\Sigma \setminus BV(\alpha_{ctrl})} = \mu|_{\Sigma \setminus BV(\alpha_{ctrl})}$. Now $\mu \models \Upsilon^+$ implies $\mu(x) = \mu(x^+) = \nu_{x^+}^{\tilde{\nu}(x)}(x^+) = \tilde{\nu}(x)$. (in other words, $\mu|_V = \tilde{\nu}|_V$). Also $\mu \models [\alpha_{\delta plant}]\varphi$. Thus, by Lemma 2, $\tilde{\nu} \models [\alpha_{\delta plant}]\varphi$ since $V \supseteq FV([\alpha_{\delta plant}]\varphi)$ and hence we have $\omega \models \varphi$ for all $(\tilde{\nu}, \omega) \in \rho(\alpha_{\delta plant})$. □

Observe that this is also true for all intermediate times $\zeta \in [0, \omega(t)]$ by the transition semantics of differential equations, where $\omega(t) \leq \varepsilon$ because $\alpha_{\delta plant}$ is bounded by ε.

Remark 2 By adding a controller execution $\langle \alpha_{ctrl} \rangle$ prior to the disturbed plant model, we synthesize prediction monitors that take the actual controller decisions into account. For safety purposes, we could just as well use a monitor definition without controller $\chi_p \equiv [\alpha_{\delta plant}]\varphi$. But that would result in a rather conservative monitor, which has to keep the CPS safe without knowledge of the actual controller decision.

3.7 Decidability and computability

One useful characteristic of ModelPlex beyond soundness is that monitor synthesis is computable, which yields a synthesis algorithm, and that the correctness of those synthesized monitors w.r.t. their specification is decidable, cf. Theorems 4 and 5.

From Lemma 5 it follows that online monitoring [18] (i.e., monitoring the last two consecutive states) is permissible. So, ModelPlex turns questions $\langle \alpha^* \rangle \phi$ into $\langle \alpha \rangle \phi$. For decidability, we first consider canonical hybrid programs α of the form $\alpha \equiv \alpha_{ctrl}; \alpha_{plant}$ where α_{ctrl} and α_{plant} are free of further nested loops. To handle differential inequalities in dℒ formulas of the form $[\alpha_{\delta plant}]\phi$, the subsequent proofs additionally use the rules for handling differential-algebraic equations [31].

Theorem 4 *(Monitor correctness is decidable) We assume canonical models of the form* $\alpha \equiv \alpha_{ctrl}; \alpha_{plant}$ *without nested loops, with solvable differential equations in* α_{plant} *and disturbed plants* $\alpha_{\delta plant}$ *with constant additive disturbance* δ *(see Definition 4) and* $F(x, x^+), \varphi, H$ *to be first-order formulas. Then, monitor correctness is decidable, i.e., the formulas* $F(x, x^+) \rightarrow \langle \alpha \rangle \Upsilon^+$, $F(x, x^+) \rightarrow \langle \alpha_{ctrl} \rangle (\Upsilon^+ \wedge H)$, *and* $F(x, x^+) \rightarrow \langle \alpha \rangle (\Upsilon^+ \wedge H \wedge [\alpha_{\delta plant}]\varphi)$ *are decidable.*

Proof From relative decidability of dℒ [33, Theorem 11] we know that sentences of dℒ (i.e., dℒ formulas without free variables) are decidable relative to an oracle for discrete loop invariants/variants and continuous differential invariants/variants. Since neither α_{ctrl} nor α_{plant} contain nested loops, we manage without an oracle for loop invariants/variants. Further, since the differential equation systems in α_{plant} are solvable, we have an effective oracle for differential invariants/variants. Let $Cl_\forall(\phi)$ denote the universal closure of dℒ formula ϕ (i.e., $Cl_\forall(\phi) \equiv \forall_{z \in FV(\phi)} z.\phi$). Note that when $\models F$ then also $\models Cl_\forall(F)$ by a standard argument.

Model monitor $F(x, x^+) \rightarrow \langle \alpha \rangle \Upsilon^+$: Follows from relative decidability of dℒ [33, Theorem 11], because $Cl_\forall(F(x, x^+) \rightarrow \langle \alpha \rangle \Upsilon^+)$ contains no free variables.
Controller monitor $F(x, x^+) \rightarrow \langle \alpha_{ctrl} \rangle (\Upsilon^+ \wedge H)$: Follows from relative decidability of dℒ [33, Theorem 11], because $Cl_\forall(F(x, x^+) \rightarrow \langle \alpha_{ctrl} \rangle (\Upsilon^+ \wedge H))$ contains no free variables.

Prediction monitor $F(x, x^+) \rightarrow \langle\alpha_{\text{ctrl}}\rangle(\Upsilon^+ \wedge H \wedge [\alpha_{\delta\text{plant}}]\varphi)$: First assume that $[\alpha_{\delta\text{plant}}]\varphi$ can be represented in a first-order formula B such that $B \rightarrow [\alpha_{\delta\text{plant}}]\varphi$. Then, by

$$(\langle\alpha\rangle\phi \wedge [\alpha](\phi \rightarrow \psi)) \rightarrow \langle\alpha\rangle(\phi \wedge \psi)$$

decidability splits into two cases:

Case $F(x, x^+) \rightarrow \langle\alpha_{\text{ctrl}}\rangle(\Upsilon^+ \wedge H \wedge B)$: follows from case $F(x, x^+) \rightarrow \langle\alpha_{\text{ctrl}}\rangle(\Upsilon^+ \wedge H)$ (controller monitor) above.

Case $(\Upsilon^+ \wedge H \wedge B) \rightarrow [\alpha_{\delta\text{plant}}]\varphi$: Since the disturbance δ in $\alpha_{\delta\text{plant}}$ is constant additive and the differential equations in α_{plant} are solvable, we have the disturbance functions $f(\theta, \delta)$ and $g(\theta, \delta)$ applied to the solution as an oracle[8] for differential invariants (i. e., the differential invariant is a pipe around the solution without disturbance). Specifically, to show $(\Upsilon^+ \wedge H \wedge B) \rightarrow [\alpha_{\delta\text{plant}}]\varphi$ by Definition 4 we have to show $(\Upsilon^+ \wedge H \wedge B) \rightarrow [x_0 := 0;\ (\theta - \delta \leq x' \leq \theta + \delta \,\&\, H \wedge x_0 \leq \varepsilon)]\varphi$. We proceed with only $(\Upsilon^+ \wedge H \wedge B) \rightarrow [x_0 := 0;\ (x' \leq \theta + \delta \,\&\, H \wedge x_0 \leq \varepsilon)]\varphi$ since the case $\theta - \delta \leq x'$ follows in a similar manner. By definition of $\alpha_{\delta\text{plant}}$ we know $0 \leq x_0$, and hence continue with $(\Upsilon^+ \wedge H \wedge B) \rightarrow [x' \leq \theta + \delta \,\&\, H \wedge 0 \leq x_0 \leq \varepsilon]\varphi$ by differential cut $0 \leq x_0$. Using the differential cut rule [31], we further supply the oracle $\text{sol}_x + \delta x_0$, where sol_x denotes the solution of $x' = \theta$ in α_{plant} and δx_0 the solution for the disturbance since δ is constant additive. This leads to two proof obligations:

Prove oracle $(\Upsilon^+ \wedge H \wedge B) \rightarrow [x' \leq \theta + \delta \,\&\, H \wedge 0 \leq x_0 \leq \varepsilon]x \leq \text{sol}_x + \delta x_0$, which by rule differential invariant [31] is valid if we can show $0 \leq x_0 \leq \varepsilon \rightarrow x' \leq \text{sol}'_x + (\delta x_0)'$ where the primed variables are replaced with the respective right-hand side of the differential equation system. From Definition 4 we know that $x'_0 = 1$ and $\delta' = 0$ and since sol_x is the solution of $x' = \theta$ in α_{plant} we further know that $\text{sol}'_x = \theta$; hence we have to show $0 \leq x_0 \leq \varepsilon \rightarrow \theta + \delta \leq \theta + \delta$, which is trivially true.

Use oracle $(\Upsilon^+ \wedge H \wedge B) \rightarrow [x' \leq \theta + \delta \,\&\, H \wedge 0 \leq x_0 \leq \varepsilon \wedge x \leq \text{sol}_x + \delta x_0]\varphi$, which by rule differential weaken [31] is valid if we can show

$$(\Upsilon^+ \wedge H \wedge B) \rightarrow \forall^\alpha ((H \wedge 0 \leq x_0 \leq \varepsilon \wedge x \leq \text{sol}_x + \delta x_0) \rightarrow \varphi)$$

where $\forall^\alpha$ denotes the universal closure w.r.t. α, i. e., $\forall x$. But since $B \rightarrow [\alpha_{\delta\text{plant}}]\varphi$ is valid, this is provable by quantifier elimination. Furthermore, we cannot get a better result than differential weaken, because the evolution domain constraint contains the oracle's answer for the differential equation system, which characterizes exactly the reachable set of the differential equation system.

We conclude that the oracle is proven correct and its usage is decidable.

It remains to show that $[\alpha_{\delta\text{plant}}]\varphi$ can be represented in a first-order formula B such that $B \rightarrow [\alpha_{\delta\text{plant}}]\varphi$. We know from Lemma 7 that any fluctuating disturbance can be approximated by its mean disturbance throughout. So for all fluctuating disturbances in $[-\delta, \delta]$ we have a corresponding constant additive mean disturbance from $[-\delta, \delta]$, which yields solvable differential equations. Hence, there exists a first-order formula B such that $B \rightarrow [\alpha_{\delta\text{plant}}]\varphi$ is valid. For the constant additive case, there even is a first-order formula B that is equivalent to $[\alpha_{\delta\text{plant}}]\varphi$, because every constant additive disturbance can be replaced equivalently by a mean disturbance using the mean-value theorem for the disturbance as a (continuous!) function of time [30]. Consequently, the

[8] By design, the disturbed plant $\alpha_{\delta\text{plant}}$ also includes a clock x_0, so the oracle additionally includes the trivial differential invariant $x_0 \geq 0$.

above cut to add B is possible if and only if the monitor χ_p is correct, leading to a decision procedure. □

For computability, we start with a theoretical proof on the basis of decidability, before we give a constructive proof, which is more useful in practice.

Theorem 5 *(Monitor synthesis is computable) We assume canonical models of the form* $\alpha \equiv \alpha_{ctrl}; \alpha_{plant}$ *without nested loops, with solvable differential equations in* α_{plant} *and disturbed plants* $\alpha_{\delta plant}$ *with constant additive disturbance* δ *(see Definition 4). Then, monitor synthesis is computable, i. e., the functions* $synth_m : \langle\alpha\rangle\Upsilon^+ \mapsto F(x, x^+)$, $synth_c : \langle\alpha_{ctrl}\rangle(\Upsilon^+ \wedge H) \mapsto F(x, x^+)$, *and* $synth_p : \langle\alpha_{ctrl}\rangle(\Upsilon^+ \wedge H \wedge [\alpha_{\delta plant}]\varphi) \mapsto F(x, x^+)$ *are computable.*

Proof Follows immediately from Theorem 4 with recursive enumeration of monitors. □

We give a constructive proof of Theorem 5. The proof is based on the observation that, except for loop and differential invariants/variants, rule application in the $\mathsf{d}\mathscr{L}$ calculus is deterministic: from [31, Theorem 2.4] we know that, relative to an oracle for first-order invariants and variants, the $\mathsf{d}\mathscr{L}$ calculus gives a semidecision-procedure for $\mathsf{d}\mathscr{L}$ formulas with differential equations having first-order definable flows.

Proof For the sake of a contradiction, suppose that monitor synthesis stopped with some open sequent not being a first-order quantifier-free formula. Then, by [31, Theorem 2.4] the open sequent either contains a hybrid program with nondeterministic repetition or a differential equation at top level, or it is not quantifier-free. But this contradicts our assumption that both α_{ctrl} and α_{plant} are free from loops and that the differential equations are solvable and disturbance is constant, in which case for

Model monitor synthesis χ_m: the solution rule $\langle' \rangle$ would make progress, because the differential equations in α_{plant} are solvable; and for
Prediction monitor synthesis χ_p: the disturbance functions $f(\theta, \delta)$ and $g(\theta, \delta)$ applied to the solution provide differential invariants (see proof of Theorem 4) so that the differential cut rule, the differential invariant rule, and the differential weakening rule [31] would make progress.

In the case of the open sequent not being quantifier-free, the quantifier elimination rule QE would be applicable and turn the formula including quantifiers into an equivalent quantifier-free formula. Hence, the open sequent neither contains nondeterministic repetition, nor a differential equation, nor a quantifier. Thus we conclude that the open sequent is a first-order quantifier-free formula. □

3.8 A proof tactic for automatic monitor synthesis

Based on the decidability and computability results above, this section explains how to implement ModelPlex monitor synthesis (Algorithm 1) as an automatic proof tactic for correct-by-construction monitor synthesis. This proof tactic is formulated in the tactic language of our theorem prover KeYmaera X [14]. KeYmaera X features a small soundness-critical core for axiomatic reasoning. On top of that core, tactics steer the proof search: *axiomatic tactics* constitute the most basic constructs of a proof, while *tactic combinators* (e. g., sequential tactic execution) are a language to combine tactics into more powerful proof procedures. The tactic language of KeYmaera X provides operators for sequential tactic composition (;), tactic repetition (*), optional execution (?), and alternatives (|) to combines basic $\mathsf{d}\mathscr{L}$ tactics, see [14].

For ModelPlex, we combine propositional axiomatic tactics with tactics for handling hybrid programs into a single tactic called *synthesize*, which performs the steps of Algorithm 1 *in place* such that the monitor is synthesized on a single proof branch by successively transforming the model. The synthesize tactic finds modalities with hybrid programs, and uses contextual equivalence rewriting to replace these modalities in place while retaining a proof of correctness of those transformations.

$$synthesize \equiv locate(prepare)^*;\ locate\,(rewriteHP)^*\,;\ local\ QE?;\ (\exists\sigma)^* \tag{6}$$

$$prepare \equiv \begin{cases} (\langle;\rangle) \mid (\cup) \mid (\langle*\rangle\ \text{elim}) & \text{model monitor} \\ (\langle;\rangle) \mid (\cup) \mid (\langle*\rangle\ \text{elim}) \mid (\langle'\rangle\ \text{skip}) & \text{controller monitor} \end{cases} \tag{7}$$

$$rewriteHP \equiv (\langle:=\rangle) \mid (\langle:=*\rangle) \mid (\langle:=\rangle\ \text{eq}) \mid (\langle?\rangle) \mid (\langle'\rangle\ \text{solve}) \tag{8}$$

The synthesize tactic operates on a specification conjecture $\langle\alpha^*\rangle\Upsilon^+$. It combines tactic selection as in a regular expression with search, so that formulas are turned into axioms step-by-step (*backwards search*). Synthesize starts with *prepare*, which determines whether to synthesize a controller monitor (unwinds loops and skips differential equations) or a model monitor (unwinds loops). Then, it repeats rewriting hybrid programs until none of the hybrid program tactics is applicable anymore, indicated by $locate(rewriteHP)^*$. Note, that the synthesize tactic does not need to instantiate existential quantifiers at intermediate steps, since it can continue rewriting inside existential quantifiers. After rewriting hybrid programs is done, an optional local quantifier elimination step is made, cf. (6), in case that any universal quantifiers remained from the ODE in the innermost sub-formula, followed by instantiating the existential quantifiers using $\exists\sigma$.

At the heart of the synthesize tactic is *locate*, which searches for the topmost formula that includes a hybrid program (i. e., a diamond modality) and chooses the appropriate tactic to reduce that program. The proof search itself is backward in sequent-style, which starts from the monitor specification conjecture and searches for steps that transform the conjecture gradually into axioms. This tactic seems like a natural way of synthesizing monitors, since it starts from the conjecture and repeatedly applies proof steps until no more progress can be made (i. e., no more steps are applicable). However, repeated search for the topmost hybrid program operator incurs considerable computation time overhead, as we will see in Sect. 4.

To avoid repeated search, we provide another tactic using a *forward chase*. The forward chase uses proof-as-computation and is based on unification and recursive forward application of axioms, which allows us to construct a proof computationally from axioms until we reach the monitor specification conjecture. Each step of the recursive computation knows the position where to apply the subsequent step, so that no search is necessary.

4 Evaluation

4.1 Monitor synthesis

We developed two software prototypes: A *sequent-style synthesis prototype* uses KeYmaera 3 [37] and Mathematica. It uses Mathematica to simplify redundant monitor conditions after synthesizing the monitor in KeYmaera 3, and therefore must recheck the final monitor for

Table 2 Case study overview

Case study	Characteristics	Dim.	Proof steps	Branches
Water tank	1 control branch, solvable ODE	5	38	4
Cruise control [20]	3 control branches,solvable ODE	11	969	124
Speed limit [25]	6 control branches,solvable ODE	9	410	30
ETCS safety [38]	8 control branches,solvable ODE	16	193	10
Robot [26]	3 control branches,non-solvable ODE	14	3350	225

correctness. An *axiomatic-style prototype* is implemented as a tactic in KeYmaera X [14], which generates controller and model monitors fully automatically and avoids branching by operating on sub-formulas in a single sequent. The axiomatic-style prototype synthesizes correct-by-construction monitors and produces a proof of correctness during the synthesis without the need to recheck.

To evaluate our method, we synthesize monitors for prior case studies of nondeterministic hybrid models of autonomous cars, train control systems, and robots (adaptive cruise control [20], intelligent speed adaptation [25], the European train control system [38], and ground robot collision avoidance [26]), see Table 2. For the model, we list the dimension in terms of the number of function symbols and state variables, as well as the size of the safety proof for proving (2), i. e., number of proof steps and the number of proof branches. The safety proofs of Formula (2) transfer from KeYmaera 3 and were not repeated in KeYmaera X.

Table 3 summarizes the evaluation results. The main result is the completely automatic, correct-by-construction synthesis in KeYmaera X with a single open branch on which the monitor is being synthesized. The monitor sizes in KeYmaera X are usually smaller than

Table 3 Monitor synthesis case studies

	Case Study	Dim.	Manual/steps (Open/branches)				Rechecking		Unsimpl.	Size
			with Opt. 1		without Opt. 1		(branches)			
	Axiomatic-style synthesis									
χ_m	Water tank	3	**0**/1698	(**1**/344)	**0**/1858	(**1**/377)				58
χ_c	Water tank	3	**0**/157	(**1**/32)	see left					12
	Cruise control	7	**0**/759	(**1**/189)	**0**/809	(**1**/204)				78
	Robot	11	**0**/6425	(**1**/2730)	see left					108
	Speed limit	6	**0**/7312	(**1**/2835)	see left					**163**
	Sequent-style synthesis									
χ_m	Water tank	3	12/16	(2/2)	3/20	(2/2)	64	(5)	$\approx\times 2$	32
	Cruise control	7	98/133	(13/13)	52/597	(21/21)	19514	(1058)	1111	1111
	Speed limit	6	335/487	(32/32)	648/5016	(126/126)	64311	(2294)	19850	19850
χ_c	Water tank	1	8/12	(2/2)	**0**/14	(2/2)	40	(3)	$\approx\times 2$	20
	Cruise control	7	48/83	(13/13)	**0**/518	(106/106)	5840	(676)	$\approx\times 10$	84
	Robot	11	68/98	(10/10)	**0**/1210	(196/196)	26166	(2854)	$\approx\times 10$	121
	Speed limit	6	247/377	(32/32)	N/A		226543	(31832)	2452	2452
	ETCS safety	13	114/156	(13/16)	0/359	(31/37)	16770	(869)	$\approx\times 10$	153
χ_p	Water tank	1	31/135	(4/10)	N/A		307	(12)	$\approx\times 2$	43

http://www.cs.cmu.edu/~smitsch/resource/modelplex_fmsd_study.zip

Table 4 Monitor synthesis duration

Case study	Axiomatic-style		Sequent-style	
	Search	Chase	Synth.	+Check
Water tank	0.9	0.3	1.3	4.9
Cruise control [20]	22.9	3.3	12.7	>1,000
Robot [26]	72.1	23.4	23.5	>1,000
Speed limit [25]	30.4	2.1	0.6[a]	204.9

[a] Synthesis with Opt.1, not counting for manual interaction

those of KeYmaera 3, because the structure is preserved better so no external simplification is needed, cf. last column "unsimplified". Without external simplification, very similar conditions with only small deviations are repeated on each open branch, For example, the controller monitor sizes listed the sequent-style synthesis need to be multiplied roughly by the number of open branches, in order to get the monitor size before external simplification.

A detailed analysis follows in subsequent paragraphs below. For the monitor, we list the dimension of the monitor specification in terms of the number of variables, compare the number of manual steps among all steps and branches left open among all branches when deriving the monitor with or without Opt. 1, as well as the number of steps when rechecking monitor correctness. Finally, we list the monitor size in terms of the number of arithmetic, comparison, and logical operators in the monitor formula. The number of proof steps of KeYmaera 3 and KeYmaera *X* are not directly comparable, because both implement different calculi. KeYmaera X leads to more but simpler proof steps.

Performance Analysis We analyze the computation time for deriving controller monitors fully automatically in the axiomatic-style synthesis technique, comparing both the backward tactic and forward chase implementations introduced in Section 3.8 above. The computation time measurements were taken on a 2.4 GHz Intel Core i7 with 16GB of memory, averaged over 20 runs. Table 4 summarizes the performance measurements for the axiomatic-style synthesis in KeYmaera X and the sequent-style synthesis in KeYmaera 3. Unsurprisingly, the repeated search for applicable positions in the backward tactic results in a considerable computation time overhead, when compared to the forward chase. Additional performance gains in the forward chase are rooted in (i) its ability to largely use derived axioms, which need only be proven once during synthesis (instead of repeatedly on each occurrence, as in the backward tactic); and (ii) its ability to postpone assignment processing and thus avoid intermediate stuttering assignments, which are necessary for successful uniform substitution [35], but result in additional proof steps if performed early.

For the sequent-style synthesis technique we list the time needed to perform the fully automated steps without Opt. 1 in KeYmaera. The raw synthesis times are comparable to those in the chase-based axiomatic-style synthesis, because the sequent-style technique always operates on the top-level operator and does not need search. Recall, however, that in the sequent-style synthesis technique the monitors are simplified with an unverified external procedure and, therefore, need to be re-checked for correctness in KeYmaera. This check needs considerable time, as listed in the last column of Table 4.

KeYmaera X The axiomatic-style synthesis prototype supports proof search steering with fine-grained tactics, and applies tactics in-place on sub-formulas, without branching on top-level first. As a result, synthesis both with and without Opt. 1 is fully automatic and avoids redundancies in monitor conditions. The reasoning style of KeYmaera X, as illustrated in

Proof 3, uses frequent cuts to collect all monitoring conditions in a single open branch, which results in a larger overall number of branches than in sequent-style synthesis. The important characteristic is that these side branches all close, so that only a single branch remains open. This means that synthesis does not require untrusted procedures to simplify monitoring conditions that were duplicated over multiple branches, which also entails that the final rechecking of the monitor is not required, see column "proof steps (branches)". Having only one branch and operating on sub-formulas also means that Opt. 1 does not need to be executed at intermediate stages in the synthesis process. Remaining existential quantifiers can be instantiated once at the end of the synthesis process, so that synthesis with and without Opt. 1 become identical.

KeYmaera X, however, is still in an early development stage and, so far, does not support differential inequalities and arbitrary differential equations in diamond modalities, so we cannot evaluate prediction monitor and model monitor synthesis fully. As development progress continues, these restrictions will diminish and we will analyze the model monitor and prediction monitor case studies with the axiomatic-style synthesis prototype once available.

KeYmaera 3 In the sequent-style synthesis prototype we support model monitor and prediction monitor synthesis for a wider range of systems, albeit at the cost of significantly increased manual interaction: for example, Opt. 1 has to be applied manually, since KeYmaera 3 does not provide sufficiently fine-grained steering of its automated proof search. Since optimization occurs after non-deterministic assignments and differential equations (i. e., in the middle of a proof), most of the synthesis process is user-guided as a consequence. For controller monitors, the sequent-style synthesis prototype without Opt. 1 is fully automatic (see number of manual steps in column "without Opt. 1" in lines 4-7, marked χ_c). In full automation, however, the proof search of KeYmaera 3 results in increased branching, since propositional steps during proofs are usually cheap (see number of branches in column "without Opt. 1"). As a consequence, even though the relative number of manual proof steps is reduced, the massive branching of the automatic proof search implies that in absolute terms more manual steps might be necessary than in the completely manual process (see number of manual steps in line 3, Speed limit case study, where local quantifier elimination after solving ODEs is performed manually). This can be avoided with fine-grained tactic support, as is achieved in the axiomatic-style synthesis prototype.

Although the number of steps and open branches differ significantly between manual interaction for Opt. 1 and automated synthesis, the synthesized monitors are logically equivalent. But applying Opt. 1 usually results in structurally simpler monitors, because the conjunction over a smaller number of open branches (cf. Table 3) can still be simplified automatically. The model monitors for cruise control and speed limit control are significantly larger than the other monitors, because their size already prevents automated simplification by Mathematica. Here, the axiomatic-style synthesis approach is expected to provide significant advantage, since it does not duplicate conditions over many branches and, thus, computes small monitors even without further simplification.

4.2 Model simulation with monitoring

We tested the ModelPlex monitors with a simulation in Mathematica[9] on hybrid system models of the water tank example used throughout this article.

[9] http://www.wolfram.com/mathematica.

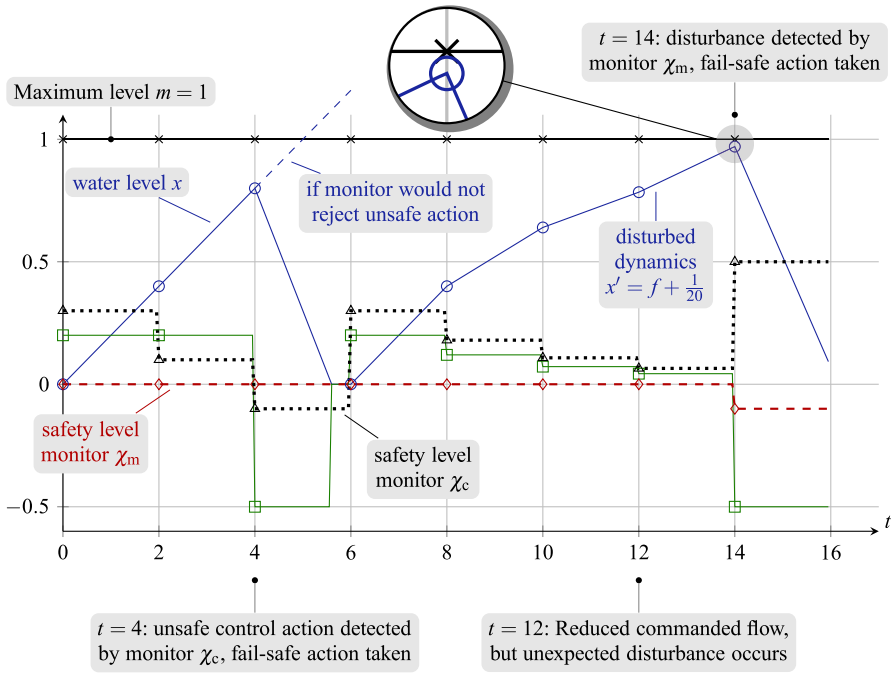


Fig. 11 Water tank simulation with monitor illustration, —×— is maximum level (m), —○— is current level (x), —□— is commanded flow (f), - ◇ - is the output of monitor χ_m for the complete model, and ·· △ ·· is the output of monitor χ_c for the controller

To illustrate the behavior of the water tank model with a fallback controller, we created two monitors: Monitor χ_m validates the complete model (as in the examples throughout this article) and is executed at the beginning of each control cycle (before the controller runs). Monitor χ_c validates only the controller of the model α^* (compares prior and posterior state of $f := *;\ ?-1 \leq f \leq \frac{m-x}{\varepsilon}$) and is executed after the controller but before control actions are issued. Thus, monitor χ_c resembles conventional runtime verification approaches, which do not check CPS behavior for compliance with the complete hybrid model. This way, we detect unexpected deviations from the model at the beginning of each control cycle, while we detect unsafe control actions immediately before they are taken. With only monitor χ_m in place we would require an additional control cycle to detect unsafe control actions[10], whereas with only monitor χ_c in place we would miss deviations from the model.

Figure 11 shows a plot of the variable traces of one simulation run. In the simulation, we ran the controller every 2 s ($\varepsilon = 2$ s, indicated by the grid for the abscissa and the marks on sensor and actuator plots). The controller was set to adjust flow to $\frac{5(m-x_0)}{2\varepsilon} = \frac{5}{2}$ for the first three controller cycles, which is unsafe on the third controller cycle. Monitor B immediately detects this violation at $t = 4$, because on the third controller cycle setting $f = \frac{5}{2}$ violates $f \leq \frac{m-x_1}{\varepsilon}$. The fail-safe action at $t = 4$ drains the tank and, after that, normal operation continues until $t = 12$. Unexpected disturbance $x' = f + \frac{1}{20}$ occurs throughout $t = [12, 14]$, which is detected by monitor χ_m. Note, that such a deviation would

[10] We could run monitor χ_m in place of χ_c to achieve the same effect. But monitor χ_m implements a more complicated formula, which is unnecessary when only the controller output should be validated.

remain undetected with conventional approaches (monitor χ_c is completely unaware of the deviation). In this simulation run, the disturbance is small enough to let the fail-safe action at $t = 14$ keep the water tank in a safe state.

5 Related work

Runtime verification and monitoring for finite state discrete systems has received significant attention (e. g., [9,16,23]). Other approaches monitor continuous-time signals (e. g., [11,28]). We focus on hybrid systems models of CPS to combine both.

Several tools for formal verification of hybrid systems are actively developed (e. g., SpaceEx [13], dReal [15], extended NuSMV/MathSat [6]). For monitor synthesis, however, ModelPlex crucially needs the rewriting capabilities and flexibility of (nested) $[\alpha]$ and $\langle\alpha\rangle$ modalities in dℒ [31] and KeYmaera [37]; it is thus an interesting question for future work if other tools could be adapted to ModelPlex.

Runtime verification is the problem of checking whether or not a trace produced by a program satisfies a particular formula (cf. [18]). In [44], a method for runtime verification of LTL formulas on abstractions of concrete traces of a flight data recorder is presented. The RV system for Java programs [22] predicts execution traces from actual traces to find concurrency errors offline (e. g., race conditions) even if the actual trace did not exhibit the error. We, instead, use prediction on the basis of disturbed plant models for *hybrid systems* at runtime to ensure safety for future behavior of the system and switch to a fail-safe fallback controller if necessary. Adaptive runtime verification [4] uses state estimation to reduce monitoring overhead by sampling while still maintaining accuracy with Hidden Markov Models, or more recently, particle filtering [17] to fill the sampling gaps. The authors present interesting ideas for managing the overhead of runtime monitoring, which could be beneficial to transfer into the hybrid systems world. The approach, however, focuses purely on the discrete part of CPS.

The Simplex architecture [39] (and related approaches, e. g., [1,3,19]) is a control system principle to switch between a highly reliable and an experimental controller at runtime. Highly reliable control modules are assumed to be verified with some other approach. Simplex focuses on switching when timing faults or violation of controller specification occur. Our method complements Simplex in that *(i)* it checks whether or not the current system execution fits the entire model, not just the controller; *(ii)* it systematically derives provably correct monitors for hybrid systems; *(iii)* it uses prediction to guarantee safety for future behavior of the system.

Further approaches with interesting insights on combined verification and monitor or controller synthesis for discrete systems include, for instance, [2,12].

Although the related approaches based on offline verification derive monitors and switching conditions from models, none of them validates whether or not the model is adequate for the current execution. Thus, they are vulnerable to deviation between the real world and the model. In summary, this article addresses safety at runtime as follows:

- Unlike [39], who focus on timing faults and specification violations, we propose a systematic principle to derive monitors that react to any deviation from the model.
- Unlike [4,17,19,22], who focus on the discrete aspects of CPS, we use hybrid system models with differential equations to address controller and plant.
- Unlike [19,39], who assume that fail-safe controllers have been verified with some other approach and do not synthesize code, we can use the same technical approach (dℒ) for verifying controllers and synthesizing provably correct monitors.

– ModelPlex combines the leight-weight monitors and runtime compliance of online runtime verification with the design time analysis of offline verification.
– ModelPlex synthesizes provably correct monitors, certified by a theorem prover.
– To the best of our knowledge, our approach is the first to guarantee that verification results about a hybrid systems model transfer to a particular execution of the system by verified runtime validation. We detect deviation from the verified model when it first occurs and, given bounds, can guarantee safety with fail-safe fallback. Other approaches (e. g., [3,19,39]) assume the system perfectly complies with the model.

6 Conclusion

ModelPlex is a principle to build and verify high-assurance controllers for safety-critical computerized systems that interact physically with their environment. It guarantees that verification results about CPS models transfer to the real system by safeguarding against deviations from the verified model. Monitors created by ModelPlex are provably correct and check at runtime whether or not the actual behavior of a CPS complies with the verified model and its assumptions. Upon noncompliance, ModelPlex initiates fail-safe fallback strategies. In order to initiate those strategies early enough, ModelPlex uses prediction on the basis of disturbed plant models to check safety for the next control cycle. This way, ModelPlex ensures that verification results about a model of a CPS transfer to the actual system behavior at runtime.

The new axiomatic-style monitor synthesis performs monitor construction in place, which enables correct-by-construction synthesis entirely within the theorem prover, constructing a proof as evidence of the correctness of the monitor. The axiomatic-style synthesis retains efficiency using contextual rewriting in a uniform substitution calculus for differential dynamic logic. It also preserves structure, leading to smaller monitor sizes.

Future research directions include extending ModelPlex with advanced $\mathsf{d}\mathscr{L}$ proof rules for differential equations [33,41], so that we not only synthesize prediction monitors from differential equations without polynomial solutions, but also model monitors. An interesting question for certification purposes is end-to-end verification from the model to the final machine code, which this article reduces to the problem of a verified translation from the monitor formula to the monitor code. This last step is conceptually straightforward but technically nontrivial in languages like C.

Acknowledgments Open access funding provided by Johannes Kepler University Linz. This material is based on research sponsored by DARPA under agreement number DARPA FA8750-12-2-0291. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The research received funding from NSF, grant agreement CNS-1054246. The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement no. PIOF-GA-2012-328378.

Appendix 1: Running example: water tank

The running example for this article is a simple water tank that will be used to illustrate the concepts throughout. The water level in the tank is controlled by a digital controller that can periodically adjust flow into and from the tank by adjusting two valves. Every time the controller decides on adjusting the flow, it measures the water level through a sensor (i. e., it samples the water level). As a safety condition, we want the water tank to never overflow: any control decision of the controller must be such that the water level stays within 0 and a maximum water level m at all times.

For proving safety, we model this example as a hybrid system model in (9).

$$\underbrace{0 \leq x \leq m \wedge \varepsilon > 0}_{\phi} \rightarrow \Big[\Big(f := *;\ ?\Big(-1 \leq f \leq \frac{m-x}{\varepsilon}\Big);\ t := 0;\ \{x' = f,\ t' = 1 \,\&\, x \geq 0 \wedge t \leq \varepsilon\}\Big)^*\Big] \overbrace{(0 \leq x \leq m)}^{\psi} \tag{9}$$

The water tank has a current water level x and a maximum water level m. The water tank controller, which runs at least every ε time units, nondeterministically chooses any flow f (written $f := *$) between a maximum outflow -1 and a maximum inflow $\frac{m-x}{\varepsilon}$ (by the subsequent test $?\big(-1 \leq f \leq \frac{m-x}{\varepsilon}\big)$). Note, that when the tank is empty ($x = 0$) and the controller still chooses a negative flow $f < 0$ as permitted by the test $?\big(-1 \leq f \leq \frac{m-x}{\varepsilon}\big)$, the evolution domain constraint $x \geq 0$ in the ODE, which models a physical constraint that water level cannot be negative, will abort immediately. As a result, only non-negative values for f will make progress in case the tank is empty. Choosing flow directly simplifies the behavior of the actual water tank implementation (a single flow value models two valves).[11] The water level in the tank evolves according to the idealized differential equation $x' = f,\ t' = 1 \,\&\, x \geq 0 \wedge t \leq \varepsilon$. Besides the water level changing according to the flow $x' = f$, the differential equation system includes time $t' = 1$ to model controller periodicity ($t \leq \varepsilon$). This considerably simplifies water flow models (e. g., it neglects the influence of water level on flow, and flow disturbance in pipes). The first conjunct $x \geq 0$ in the evolution domain models a physical constraint that the water level can never be below zero (otherwise, the differential equation would include negative water content in the tank below zero on negative flow). The second conjunct $t \leq \varepsilon$ models the sampling period of our controller, that is it allows the ODE to be followed only for at most ε time before interrupting it for a control decision. Note, that through $t \leq \varepsilon$ the sampling period does not need to be the same on every control cycle, nor does it need to be exactly ε time. This water tank never overflows, as witnessed by a proof for the $\mathsf{d}\mathscr{L}$ formula (1).

However, since we made approximations when modeling the controller and the physics, and since failures and other deviations may occur in reality (e. g., a valve could fail), we cannot simply transfer this safety proof to the real system.

First, since failures may occur we need to monitor actual evolution, such as that the actual water level corresponds to the level expected by the chosen valve positions and the actual time passed between controller executions does not exceed the modeled sampling period. The monitor needs to allow some slack around the expected water level to compensate for the

[11] In this example, this simplification is admittedly somewhat artificial but reminiscent of aspects of more general systems. In more complicated systems, such as adaptive cruise control, a controller needs to aim for passenger comfort, high fuel economy, and other secondary goals besides ensuring safety, so focusing on the safety-relevant features greatly fosters safety verification.

neglected physical phenomena. Sections 3.2 and 3.5 describe how to synthesize such model monitors automatically. Second, the controller implementation differs from the model, so we need to check that the implemented controller only chooses flows f that satisfy $-1 \leq f \leq \frac{m-x}{\varepsilon}$. Section 3.4 describes how to synthesize such controller monitors automatically. Finally, we can additionally monitor controller decisions for the expected real-world effect, since the hybrid system model contains a model of the physics of the water tank. Section 3.6 describes how to synthesize such prediction monitors automatically. The controller in the model, which is verified to be safe, gives us a fail-safe action that we can execute in place of the unverified controller implementation when one of the monitors is not satisfied.

Appendix 2: Water tank monitor specification conjecture analysis

$$(\wedge\mathrm{r})\ \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta} \qquad (\mathrm{Wr})\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \phi, \Delta} \qquad (\mathrm{QE})\ \frac{\mathrm{QE}(\phi)}{\phi}\ [1]$$

$$(\langle;\rangle)\ \langle\alpha;\beta\rangle\phi \leftrightarrow \langle\alpha\rangle\langle\beta\rangle\phi \qquad (\langle?\rangle)\ \langle?H\rangle\psi \leftrightarrow (H \wedge \psi) \qquad (\langle:=\rangle)\ \langle x := \theta\rangle\phi(x) \leftrightarrow \phi(\theta)$$

$$(\langle:=\rangle\ \mathrm{eq})\ \langle x := \theta\rangle\phi(x) \leftrightarrow \forall x = \theta\phi(x) \qquad (\langle:=*\rangle)\ \langle x := *\rangle\phi(x) \leftrightarrow \exists x\phi(x)$$

$$(\langle'\rangle)\ \big(y(0) = x \wedge [T' = 1]y(T)' = \theta(y(T))\big) \rightarrow \Big(\langle x' = \theta\ \&\ H\rangle\phi \leftrightarrow \exists T{\geq}0\ \big((\forall 0{\leq}\tilde{t}{\leq}T\ \langle x := y(\tilde{t})\rangle H) \wedge \langle x := y(T)\rangle\phi\big)\Big)\ [2]$$

[1] iff $\phi \equiv \mathrm{QE}(\phi)$, ϕ is a first-order real arithmetic formula, $\mathrm{QE}(\phi)$ is a quantifier-free formula

[2] T and $\tilde{t}$ are fresh logical variables

$$\begin{array}{ll}
 & \vdash \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge (\ldots)\big) \\
\text{Opt. 1} & \vdash \exists f\ \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \exists t = 0\, \exists \tilde{t} \geq 0\ (\ldots)\big) \\
\text{QE} & \vdash \exists f\ \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \exists t = 0\, \exists \tilde{t} \geq 0\ \big(\forall 0 \leq \tilde{s} \leq \tilde{t}\ (f\tilde{s} + x \geq 0 \wedge \tilde{s} + t \leq \varepsilon) \\
 & \qquad \wedge (f\tilde{t} + x = x^+ \wedge f = f^+ \wedge \tilde{t} + t = t^+)\big)\big) \\
\langle:=\rangle & \vdash \exists f\ \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \exists t = 0\, \exists \tilde{t} \geq 0\ \big(\forall 0 \leq \tilde{s} \leq \tilde{t}\ (f\tilde{s} + x \geq 0 \wedge \tilde{s} + t \leq \varepsilon) \\
 & \qquad \wedge \langle t := \tilde{t} + t\rangle(f\tilde{t} + x = x^+ \wedge f = f^+ \wedge t = t^+)\big)\big) \\
\langle:=\rangle & \vdash \exists f\ \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \exists t = 0\, \exists \tilde{t} \geq 0\ \big(\forall 0 \leq \tilde{s} \leq \tilde{t}\ (f\tilde{s} + x \geq 0 \wedge \tilde{s} + t \leq \varepsilon) \\
 & \qquad \wedge \langle x := f\tilde{t} + x\rangle\langle t := \tilde{t} + t\rangle(x = x^+ \wedge f = f^+ \wedge t = t^+)\big)\big) \\
\langle:=\rangle & \vdash \exists f\ \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \exists t = 0\, \exists \tilde{t} \geq 0\ \big(\forall 0 \leq \tilde{s} \leq \tilde{t}\ \langle t := \tilde{s} + t\rangle(f\tilde{s} + x \geq 0 \wedge t \leq \varepsilon) \\
 & \qquad \wedge \langle x := f\tilde{t} + x\rangle\langle t := \tilde{t} + t\rangle\Upsilon^+\big)\big) \\
\langle:=\rangle & \vdash \exists f\ \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \exists t = 0\, \exists \tilde{t} \geq 0\ \big(\forall 0 \leq \tilde{s} \leq \tilde{t}\ \langle x := f\tilde{s} + x\rangle\langle t := \tilde{s} + t\rangle(x \geq 0 \wedge t \leq \varepsilon) \\
 & \qquad \wedge \langle x := f\tilde{t} + x\rangle\langle t := \tilde{t} + t\rangle\Upsilon^+\big)\big) \\
\langle'\rangle & \vdash \exists f\ \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \exists t = 0\, \langle x' = f, t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon\rangle\Upsilon^+\big) \\
\langle:=\rangle\ \mathrm{eq} & \vdash \exists f\ \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \langle t := 0\rangle\langle x' = f, t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon\rangle\Upsilon^+\big) \\
\langle;\rangle & \vdash \exists f\ \big(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \langle \mathit{plant}\rangle\Upsilon^+\big) \\
\langle?\rangle & \vdash \exists f\, \langle ? - 1 \leq f \leq \frac{m-x}{\varepsilon}\rangle\langle \mathit{plant}\rangle\Upsilon^+ \\
\langle;\rangle & \vdash \exists f\ \big(\langle ? - 1 \leq f \leq \frac{m-x}{\varepsilon}; \mathit{plant}\rangle\Upsilon^+\big) \\
\langle:=*\rangle & \vdash \langle f := *\rangle\langle ? - 1 \leq f \leq \frac{m-x}{\varepsilon}; \mathit{plant}\rangle\Upsilon^+ \\
\langle;\rangle & \vdash \langle f := *; ? - 1 \leq f \leq \frac{m-x}{\varepsilon}; \mathit{plant}\rangle\Upsilon^+
\end{array}$$

Proof 1: Analysis of the water tank monitor specification conjecture (*plant* is an abbreviation for $t := 0;\ x' = f, t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon$, Υ^+ is an abbreviation for $x = x^+ \wedge f = f^+ \wedge t = t^+$)

Appendix 3: Sequent-style monitor synthesis

This section lists the monitor synthesis approach in the sequent style of the short version [24]. Proof 2 shows a complete sequence of proof rules applied to the water tank specification conjecture of Example 5 on page 17, with $\Upsilon^+ \equiv x = x^+ \wedge f = f^+ \wedge t = t^+$.

$$(\wedge\mathrm{r})\ \frac{\Gamma \vdash \phi, \Delta \quad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \wedge \psi, \Delta} \qquad (\mathrm{Wr})\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \phi, \Delta} \qquad (\mathrm{QE})\ \frac{\mathrm{QE}(\phi)}{\phi}\ [1]$$

$$(\langle;\rangle)\ \frac{\langle\alpha\rangle\langle\beta\rangle\phi}{\langle\alpha;\beta\rangle\phi} \qquad (\langle?\rangle)\ \frac{H \wedge \psi}{\langle?H\rangle\psi} \qquad (\langle:=\rangle)\ \frac{\phi_x^\theta}{\langle x := \theta\rangle\phi} \qquad (\langle:=*\rangle)\ \frac{\exists X \langle x := X\rangle\phi}{\langle x := *\rangle\phi}\ [2]$$

$$(\langle'\rangle)\ \frac{\exists t{\geq}0 \left(\forall 0{\leq}\tilde{t}{\leq}t\ \langle x := y(\tilde{t})\rangle H \wedge \langle x := y(t)\rangle\phi\right)}{\langle x' = \theta\ \&\ H\rangle\phi}\ [3] \qquad (\exists\mathrm{r})\ \frac{\Gamma \vdash \phi(\theta), \exists x\, \phi(x), \Delta}{\Gamma \vdash \exists x\, \phi(x), \Delta}\ [4]$$

$$(\mathrm{i}\exists)\ \frac{\Gamma \vdash \exists X \left(\bigwedge_i (\Phi_i \vdash \Psi_i)\right), \Delta}{\Gamma, \Phi_1 \vdash \Psi_1, \Delta \quad \cdots \quad \Gamma, \Phi_n \vdash \Psi_n, \Delta}\ [5] \qquad (\exists\sigma)\ \frac{\phi_x^\theta}{\exists x\, (x = \theta \wedge \phi(x))}$$

[1] iff $\phi \equiv \mathrm{QE}(\phi)$, ϕ is a first-order real arithmetic formula, $\mathrm{QE}(\phi)$ is a quantifier-free formula

[2] X is a new logical variable

[3] t and $\tilde{t}$ are fresh logical variables and $\langle x := y(t)\rangle$ is the discrete assignment belonging to the solution y of the differential equation with constant symbol x as symbolic initial value.

[4] θ is an arbitrary term, often a new (existential) logical variable X.

[5] Among all open branches, free logical variable X only occurs in the branches $\Gamma, \Phi_i \vdash \Psi_i, \Delta$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash -1 \leq f^+ \leq \frac{m-x}{\varepsilon} \wedge x^+ = x + f^+t^+ \wedge t^+ \geq 0 \wedge x \geq 0}{\wedge \varepsilon \geq t^+ \geq 0 \wedge f^+t^+ + x \geq 0}\exists\sigma}{\vdash \exists F(-1 \leq F \leq \frac{m-x}{\varepsilon} \wedge F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ \geq 0}}{\wedge x \geq 0 \wedge \varepsilon \geq t^+ \geq 0 \wedge Ft^+ + x \geq 0)}}{\vdash F = f^+ \wedge x^+ = x + Ft^+ \wedge t^+ \geq 0 \wedge x \geq 0}}{\wedge \varepsilon \geq t^+ \geq 0 \wedge Ft^+ + x \geq 0}\mathrm{QE}}{\vdash \forall 0{\leq}\tilde{t}{\leq}t^+\ (x + F\tilde{t} \geq 0 \wedge \tilde{t} \leq \varepsilon) \wedge F = f^+}}{\wedge x^+ = x + Ft^+ \wedge t^+ = t^+}}{\vdash \exists T{\geq}0\big((\forall 0{\leq}\tilde{t}{\leq}T\ (x + F\tilde{t} \geq 0 \wedge \tilde{t} \leq \varepsilon))}\exists\mathrm{r},\mathrm{Wr}}{\wedge (F = f^+ \wedge x^+ = x + FT \wedge t^+ = T)\big)}}{\vdash \langle f := F; t := 0\rangle\langle x' = f, t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon\rangle\Upsilon^+}\langle'\rangle}{\vdash \langle f := F\rangle\langle t := 0; (x' = f, t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon)\rangle\Upsilon^+}\langle;\rangle,\langle:=\rangle}{\vdash \langle f := F\rangle\langle plant\rangle\Upsilon^+}\mathrm{i}\exists$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash -1 \leq F \leq \frac{m-x}{\varepsilon}}{\vdash \langle f := F\rangle - 1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \langle plant\rangle\Upsilon^+}\wedge\mathrm{r}}{\vdash \langle f := F\rangle\langle ? - 1 \leq f \leq \frac{m-x}{\varepsilon}\rangle\langle plant\rangle\Upsilon^+}\langle?\rangle}{\vdash \langle f := F\rangle\langle ? - 1 \leq f \leq \frac{m-x}{\varepsilon}; plant\rangle\Upsilon^+}\langle;\rangle}{\vdash \langle f := F\rangle\langle ? - 1 \leq f \leq \frac{m-x}{\varepsilon}; plant\rangle\Upsilon^+}\exists\mathrm{r},\mathrm{Wr}}{\vdash \exists F\langle f := F\rangle\langle ? - 1 \leq f \leq \frac{m-x}{\varepsilon}; plant\rangle\Upsilon^+}\langle:=*\rangle}{\vdash \langle f := *\rangle\langle ? - 1 \leq f \leq \frac{m-x}{\varepsilon}; plant\rangle\Upsilon^+}\langle;\rangle}{\vdash \langle f := *; ? - 1 \leq f \leq \frac{m-x}{\varepsilon}; plant\rangle\Upsilon^+}$$

Proof 2: Analysis of the water tank monitor specification conjecture (*plant* is an abbreviation for $x' = f, t' = 1\ \&\ x \geq 0 \wedge t \leq \varepsilon$).

Instantiating existential quantifiers ahead of time. The sequent-style synthesis must instantiate existential quantifiers when they are met in the synthesis process, even though not all the

information about the exact instance may be available at that time. Opt. 1 below introduces a heuristic to avoid duplicate work when instantiating existential quantifiers.

By axiom ∃inst, an existentially quantified variable can be instantiated with an arbitrary term θ, which is often a new logical variable that is implicitly existentially quantified [29].

Optimization 1 *(Instantiation trigger) If the variable is not changed in the remaining α, $x_i = x_i^+$ is in Υ^+ and x is not bound in Υ^+, then instantiate the existential quantifier by axiom ∃inst with the corresponding x_i^+ that is part of the specification conjecture (i. e., $\theta = x_i^+$), since subsequent proof steps are going to reveal $\theta = x_i^+$ anyway. This optimization is most effective in the sequent-style synthesis technique, which spreads F over many branches.*

Otherwise, we introduce a new logical variable, which may result in an existential quantifier in the monitor specification if no further constraints can be found later in the proof.

Example 10 (Instantiating existential quantifiers) Continuing Example 6, we show the proof without and with application of Opt. 1. The corresponding steps in the water tank proof use ∃inst to instantiate the resulting existential quantifier $\exists f$ (i) with a new logical variable F (without Opt. 1), and (ii) with posterior variable f^+ (with Opt. 1). The hybrid program *plant* is an abbreviation for $x' = f, t' = 1 \,\&\, x \geq 0 \wedge t \leq \varepsilon$.

$$(\exists\text{inst})\ \phi(\theta) \rightarrow \exists x\, \phi(x)$$

$$\dfrac{\vdash -1 \leq F \leq \frac{m-x}{\varepsilon} \wedge \langle plant \rangle \Upsilon^+}{\vdash \exists f \left(-1 \leq f \leq \frac{m-x}{\varepsilon} \wedge \langle plant \rangle \Upsilon^+\right)}\ \exists\text{inst} \qquad \dfrac{\vdash -1 \leq f^+ \leq \frac{m-x}{\varepsilon} \wedge \langle plant \rangle \Upsilon^+}{\vdash \ldots}\ \exists\text{inst}$$

w/o Opt. 1

with Opt. 1 (anticipate $f = f^+$ from Υ^+)

Re-introducing quantifiers In the sequent-style analysis, the fragments of the monitor are usually scattered over several branches, since many proof rules split into two or more branches, as ∧r after ⟨?⟩ in Proof 2 above. At the same time, sequent-style reasoning has the main goal to make proving properties easy (as opposed to synthesis) and thus only works on the top-level operator of a formula, not inside as in the axiomatic-style synthesis. As a result, the sequent-style prototype cannot postpone instantiating existential quantifiers until later, so it either must use Opt. 1 or instantiate with a fresh variable (e. g., F in ∃r in Proof 2 above). In the latter case, in the final step of sequent-style synthesis we can re-introduce existential quantifiers and look for additional facts that let us instantiate the quantifier with a more useful variable, see Example 11.

We use the invertible quantifier rule i∃ to re-introduce existential quantifiers for the new logical variables (universal quantifiers for function symbols, see [29] for calculus details). Often, the now quantified logical variables are discovered to be equal to one of the post-state variables later in the proof, because those variables did not change in the model after the assignment. If this is the case, we can use proof rule $\exists\sigma$ to further simplify the monitor specification by substituting the corresponding logical variable x with its equal term θ.

Example 11 (Reintroducing existential quantifiers over multiple branches) Proof 2 uses a new logical variable F for the nondeterministic flow assignment $f := *$. After further steps in the proof, the assumptions reveal additional information $F = f^+$. Thus, we re-introduce the existential quantifier over all the open branches (i∃) and substitute f^+ for F ($\exists\sigma$). The sole open sequent of this proof attempt is the monitor specification $F(x, x^+)$ of the water tank model.

$$(\mathrm{i}\exists)\quad \frac{\Gamma \vdash \exists X \left(\bigwedge_i (\Phi_i \vdash \Psi_i)\right), \Delta}{\Gamma, \Phi_1 \vdash \Psi_1, \Delta \quad \cdots \quad \Gamma, \Phi_n \vdash \Psi_n, \Delta}\ {}^{1}$$

[1] Among all open branches, free logical variable X only occurs in Φ_i, Ψ_i in the branches $\Gamma, \Phi_i \vdash \Psi_i, \Delta$

$$\exists\sigma\ \frac{\vdash -1 \le f^+ \le \frac{m-x}{\varepsilon} \wedge x^+ = x + f^+ t^+ \wedge x \ge 0 \wedge \varepsilon \ge t^+ \ge 0 \wedge f^+ t^+ + x \ge 0}{\vdash \exists F = f^+ \left(-1 \le F \le \frac{m-x}{\varepsilon} \wedge x^+ = x + F t^+ \wedge x \ge 0 \wedge \varepsilon \ge t^+ \ge 0 \wedge F t^+ + x \ge 0\right)}$$
$$\mathrm{i}\exists\ \frac{}{\vdash -1 \le F \le \frac{m-x}{\varepsilon} \wedge F = f^+ \wedge x^+ = x + F t^+ \wedge x \ge 0 \wedge \varepsilon \ge t^+ \ge 0 \wedge F t^+ + x \ge 0}$$

Appendix 4: Reasoning in KeYmaera X

In order to illustrate the underlying principle of the synthesize tactic, let us first consider a tactic $\langle{:=}\rangle$ for nondeterministic assignment. The tactic rewrites a formula of the form $\langle x := *\rangle \phi(x)$ into $\exists x\, \phi(x)$, and proves the correctness of this rewriting from the corresponding axiom $\langle x := *\rangle \phi(x) \leftrightarrow \exists x\, \phi(x)$. Proof 3 illustrates this principle in more detail. First, the desired outcome is cut in as an equivalence to the current formula. This equivalence can be shown from the axioms, cf. right-most branch. Once proven correct, propositional tactics close one direction of the equivalence (left-most branch) and transform the equivalence into the desired outcome, cf. middle branch.

$$(\wedge\mathrm{l})\ \frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \wedge \psi \vdash \Delta} \qquad (\leftrightarrow\mathrm{l})\ \frac{\Gamma, \phi \wedge \psi \vdash \Delta \quad \Gamma, \neg\phi \wedge \neg\psi \vdash \Delta}{\Gamma, \phi \leftrightarrow \psi \vdash \Delta} \qquad (\neg\mathrm{l})\ \frac{\Gamma \vdash \phi, \Delta}{\Gamma, \neg\phi \vdash \Delta}$$

$$(\mathrm{Wl})\ \frac{\Gamma \vdash \Delta}{\Gamma, \phi \vdash \Delta} \qquad (\mathrm{Wr})\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \phi, \Delta} \qquad (\mathrm{ax})\ \frac{}{\Gamma, \phi \vdash \phi, \Delta} \qquad (\mathrm{cut})\ \frac{\Gamma, \phi \vdash \Delta \quad \Gamma \vdash \phi, \Delta}{\Gamma \vdash \Delta}$$

$$\mathrm{cut}\ \frac{\leftrightarrow\mathrm{l}\ \dfrac{\wedge\mathrm{l}\ \dfrac{\mathrm{ax}\ \dfrac{*}{\langle x := *\rangle\phi, \exists x\, \phi \vdash \langle x := *\rangle\phi}}{\langle x := *\rangle\phi \wedge \exists x\, \phi \vdash \langle x := *\rangle\phi}}{\langle x := *\rangle\phi \leftrightarrow \exists x\, \phi \vdash \langle x := *\rangle\phi} \qquad \mathrm{Wl,Wr}\ \dfrac{\vdash \exists x\, \phi}{\neg\langle x := *\rangle\phi \vdash \langle x := *\rangle\phi,\ \exists x\, \phi}}{\neg\langle x := *\rangle\phi \wedge \neg\exists x\, \phi \vdash \langle x := *\rangle\phi}\ \neg\mathrm{l}$$
$$\mathrm{Wr}\ \frac{\mathrm{ax}\ \dfrac{*}{\vdash \langle x := *\rangle\phi \leftrightarrow \exists x\, \phi}}{\vdash \langle x := *\rangle\phi,\ \langle x := *\rangle\phi \leftrightarrow \exists x\, \phi}$$
$$\vdash \langle x := *\rangle\phi$$

Proof 3: The steps taken by the tactic procedure of a tactic $\langle{:=}*\rangle$ for handling nondeterministic assignment based on an axiom $\langle x := *\rangle\phi \leftrightarrow \exists x\, \phi$

References

1. Aiello AM, Berryman JF, Grohs JR, Schierman JD (2010) Run-time assurance for advanced flight-critical control systems. In: AIAA guidance, navigation and control conference, AIAA, doi:10.2514/6.2010-8041
2. Alur R, Bodík R, Juniwal G, Martin MMK, Raghothaman M, Seshia SA, Singh R, Solar-Lezama A, Torlak E, Udupa A (2013) Syntax-guided synthesis. In: FMCAD, IEEE, pp 1–17
3. Bak S, Greer A, Mitra S (2010) Hybrid cyberphysical system verification with Simplex using discrete abstractions. In: Caccamo M (ed) IEEE Real-time and embedded technology and applications symposium, IEEE Computer Society, pp 143–152
4. Bartocci E, Grosu R, Karmarkar A, Smolka SA, Stoller SD, Zadok E, Seyster J (2012) Adaptive runtime verification. In: Qadeer S, Tasiran S (eds) Runtime verification. LNCS, vol 7687. Springer, Berlin, pp 168–182

5. Blech JO, Falcone Y, Becker K (2012) Towards certified runtime verification. In: Aoki T, Taguchi K (eds) International conference on formal engineering methods. LNCS, vol 7635. Springer, Berlin, pp 494–509
6. Cimatti A, Mover S, Tonetta S (2013) SMT-based scenario verification for hybrid systems. Form Methods Syst Des 42(1):46–66
7. Collins GE, Hong H (1991) Partial cylindrical algebraic decomposition for quantifier elimination. J Symb Comput 12(3):299–328
8. Daigle MJ, Roychoudhury I, Biswas G, Koutsoukos XD, Patterson-Hine A, Poll S (2010) A comprehensive diagnosis methodology for complex hybrid systems: a case study on spacecraft power distribution systems. IEEE Trans Syst Man Cybern Part A 40(5):917–931
9. D'Angelo B, Sankaranarayanan S, Sánchez C, Robinson W, Finkbeiner B, Sipma HB, Mehrotra S, Manna Z (2005) LOLA: Runtime monitoring of synchronous systems. In: TIME, IEEE Computer Society, pp 166–174
10. Davenport JH, Heintz J (1988) Real quantifier elimination is doubly exponential. J Symb Comput 5(1–2):29–35. doi:10.1016/S0747-7171(88)80004-X
11. Donzé A, Ferrère T, Maler O (2013) Efficient robust monitoring for STL. In: Sharygina N, Veith H (eds) Computer aided verification. LNCS, vol 8044. Springer, Berlin, pp 264–279
12. Ehlers R, Finkbeiner B (2011) Monitoring realizability. In: Khurshid S, Sen K (eds) Runtime verification. LNCS, vol 7186. Springer, Berlin, pp 427–441
13. Frehse G, Guernic CL, Donzé A, Cotton S, Ray R, Lebeltel O, Ripado R, Girard A, Dang T, Maler O (2011) SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan G, Qadeer S (eds) Computer aided verification. LNCS, vol 6806. Springer, Berlin, pp 379–395
14. Fulton N, Mitsch S, Quesel J, Völp M, Platzer A (2015) Keymaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty AP, Middeldorp A (eds) Automated deduction - CADE-25 - 25th international conference on automated deduction, Berlin, 1–7 Aug 2015, Proceedings, Springer, Lecture Notes in Computer Science, vol 9195, pp 527–538, doi:10.1007/978-3-319-21401-6_36
15. Gao S, Kong S, Clarke EM (2013) dReal: an SMT solver for nonlinear theories over the reals. In: Bonacina MP (ed) International conference on automated deduction. LNCS, vol 7898. Springer, Berlin, pp 208–214
16. Havelund K, Rosu G (2004) Efficient monitoring of safety properties. STTT 6(2):158–173
17. Kalajdzic K, Bartocci E, Smolka SA, Stoller SD, Grosu R (2013) Runtime verification with particle filtering. In: Legay A, Bensalem S (eds) Runtime verification. LNCS, vol 8174. Springer, Berlin
18. Leucker M, Schallhart C (2009) A brief account of runtime verification. J Log Algebr Program 78(5):293–303
19. Liu X, Wang Q, Gopalakrishnan S, He W, Sha L, Ding H, Lee K (2008) ORTEGA: An efficient and flexible online fault tolerance architecture for real-time control systems. IEEE Trans Ind Inform 4(4):213–224
20. Loos SM, Platzer A, Nistor L (2011) Adaptive cruise control: Hybrid, distributed, and now formally verified. In: Butler M, Schulte W (eds) Formal methods, Springer, LNCS, vol 6664, doi:10.1007/978-3-642-21437-0_6
21. McIlraith SA, Biswas G, Clancy D, Gupta V (2000) Hybrid systems diagnosis. In: Lynch NA, Krogh BH (eds) Hybrid systems: computation and cpmtrol. LNCS, vol 1790. Springer, Berlin, pp 282–295
22. Meredith PO, Rosu G (2010) Runtime verification with the RV system. In: Barringer H, Falcone Y, Finkbeiner B, Havelund K, Lee I, Pace GJ, Rosu G, Sokolsky O, Tillmann N (eds) Runtime verification. LNCS, vol 6418. Springer, Berlin, pp 136–152
23. Meredith PO, Jin D, Griffith D, Chen F, Rosu G (2012) An overview of the MOP runtime verification framework. STTT 14(3):249–289
24. Mitsch S, Platzer A (2014) ModelPlex: Verified runtime validation of verified cyber-physical system models. In: Bonakdarpour B, Smolka SA (eds) Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22–25, 2014. Proceedings, Springer, Lecture Notes in Computer Science, vol 8734, pp 199–214, doi:10.1007/978-3-319-11164-3_17
25. Mitsch S, Loos SM, Platzer A (2012) Towards formal verification of freeway traffic control. In: Lu C (ed) ICCPS, IEEE, pp 171–180, doi:10.1109/ICCPS.2012.25
26. Mitsch S, Ghorbal K, Platzer A (2013) On provably safe obstacle avoidance for autonomous robotic ground vehicles. In: Fox D, Hsu D (eds) Newman P. Robotics, Science and Systems. Technische Univ., Berlin
27. Mitsch S, Quesel JD, Platzer A (2014) Refactoring, refinement, and reasoning: A logical characterization for hybrid systems. In: Jones CB, Pihlajasaari P, Sun J (eds) Formal methods, vol 8442, Springer, pp 481–496, doi:10.1007/978-3-319-06410-9_33
28. Nickovic D, Maler O (2007) AMT: A property-based monitoring tool for analog systems. In: Raskin JF, Thiagarajan PS (eds) FORMATS. LNCSSpringer, Berlin, pp 304–319
29. Platzer A (2008) Differential dynamic logic for hybrid systems. J Autom Reason 41(2):143–189. doi:10.1007/s10817-008-9103-8

30. Platzer A (2010a) Differential-algebraic dynamic logic for differential-algebraic programs. J Log Comput 20(1):309–352, 2008, doi:10.1093/logcom/exn070, advance access published on 18 Nov
31. Platzer A (2010b) Logical analysis of hybrid systems. Springer, New York. doi:10.1007/978-3-642-14509-4
32. Platzer A (2011) The structure of differential invariants and differential cut elimination. Log Methods Comput Sci 8(4):1
33. Platzer A (2012a) The complete proof theory of hybrid systems. In: LICS, IEEE, doi:10.1109/LICS.2012.64
34. Platzer A (2012b) Logics of dynamical systems. In: LICS, IEEE, pp 13–24, doi:10.1109/LICS.2012.13
35. Platzer A (2015) A uniform substitution calculus for differential dynamic logic. In: Felty AP, Middeldorp A (eds) Conference on automated deduction. LNCS, vol 9195. Springer, pp 467–481, doi:10.1007/978-3-319-21401-6_32, 1503.01981
36. Platzer A, Clarke EM (2007) The image computation problem in hybrid systems model checking. In: Bemporad A, Bicchi A, Buttazzo G (eds) Hybrid systems: computation control. LNCS, Springer. doi:10.1007/978-3-540-71493-4_37
37. Platzer A, Quesel JD (2008) KeYmaera: A hybrid theorem prover for hybrid systems. In: Armando A, Baumgartner P, Dowek G (eds) International joint conference on automated reasoning. LNCS, vol 5195. Springer, Berlin. doi:10.1007/978-3-540-71070-7_15
38. Platzer A, Quesel JD (2009) European Train Control System: A case study in formal verification. In: Breitman K, Cavalcanti A (eds) International conference on formal engineering methods. LNCS, vol 5885. Springer, Berlin. doi:10.1007/978-3-642-10373-5_13
39. Seto D, Krogh B, Sha L, Chutinan A (1998) The Simplex architecture for safe online control system upgrades. In: American control conference, pp 3504–3508, doi:10.1109/ACC.1998.703255
40. Shannon C (1949) Communication in the presence of noise. Proc IRE 37(1):10–21. doi:10.1109/JRPROC.1949.232969
41. Sogokon A, Jackson PB (2015) Direct formal verification of liveness properties in continuous and hybrid dynamical systems. In: Bjørner N, de Boer FD (eds) FM 2015: formal methods– 20th international symposium, Oslo. 24–26 June 2015. Proceedings, Springer, Lecture Notes in Computer Science, vol 9109, pp 514–531, doi:10.1007/978-3-319-19249-9_32
42. Srivastava AN, Schumann J (2013) Software health management: a necessity for safety critical systems. ISSE 9(4):219–233
43. Wang D, Yu M, Low CB, Arogeti S (2013) Model-based health monitoring of hybrid systems. Springer, New York. doi:10.1007/978-1-4614-7369-5
44. Wang S, Ayoub A, Sokolsky O, Lee I (2011) Runtime verification of traces under recording uncertainty. In: Sen K, Khurshid S (eds) Runtime verification. LNCS, Springer, Berlin, pp 442–456
45. Zhao F, Koutsoukos XD, Haussecker HW, Reich J, Cheung P (2005) Monitoring and fault diagnosis of hybrid systems. IEEE Trans Syst Man Cybern Part B 35(6):1225–1240